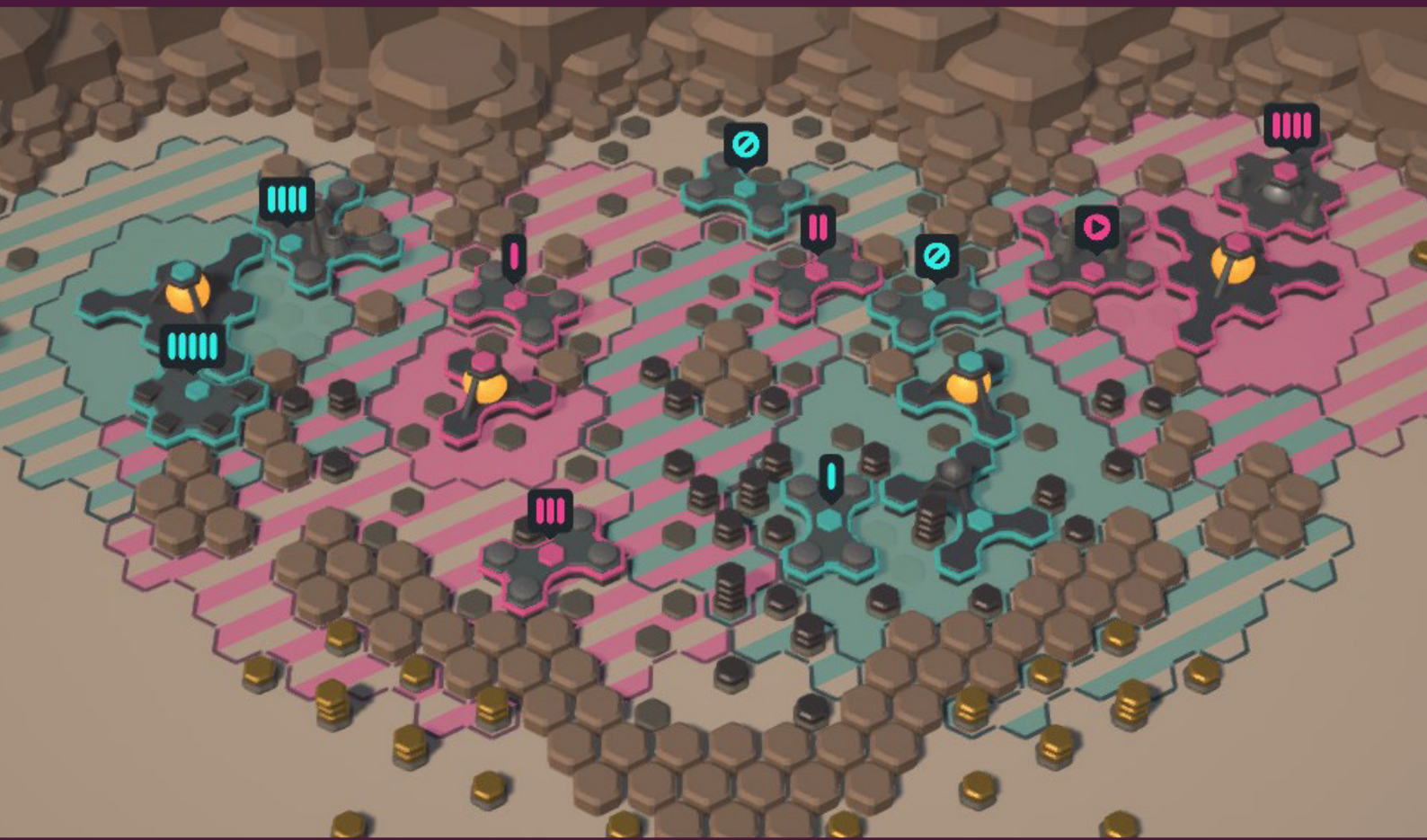


STRATEGY GAME WITH AI

Parallel Development of a Strategy Game
and AI-Opponent



Bachelor Thesis by
MILAN MROS



Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Strategy Game with AI

Parallel Development of a Strategy Game and AI-Opponent

Bachelorarbeit

Name des Studiengangs

Game Design

Fachbereich Gestaltung und Kultur

vorgelegt von:

Milan Mros

Datum: 08.06.2021

Erstgutachter: Prof. Thomas Bremer

Zweitgutachter: Prof. Jan Berger

STRATEGY GAME WITH AI

Parallel Development of a Strategy Game and AI-Opponent

Bachelor Thesis in the course of GAME DESIGN

The scope of work also contains a game system design workpiece

TABLE OF CONTENTS

1. Introduction	1
1.1 Plan and Adjustments	1
1.2 Motivation	2
1.3 Scope of Work	3
2. Development	5
2.1 Development Principles	5
2.1.1 Simplicity.	5
2.1.2 Prioritizing Gameplay Experience	6
2.1.3 Iterative Workflow.	6
2.1.4 Clear Vision.	7
2.2 Ideation	8
2.2.1 Ideation Guidelines	8
2.2.2 Buildings and Area Control	8
2.2.3 Resource and Economy	12
2.2.4 Building Cycle	13
2.2.5 Game Structure	14
2.2.6 Design Pillars.	15
2.2.7 Narrative Concept	17
2.3 Fundamental Technical Systems	18
2.3.1 Hexagonal Map System	18
2.3.2 Generic Player Architecture	21
2.3.3 Human Player Interfaces	22

2.4 Version 1 – Minimum Viable Prototype	25
2.4.1 Minimal Economy	25
2.4.2 Paper Prototype	28
2.4.3 Prototype Digitalization	31
2.4.4 Version 1 – Evaluation	31
2.4.5 Workflow Evaluation and Outlook	33
2.5 Version 2 – Core Prototype	34
2.5.1 Game Design	34
2.5.2 Visual Communication	37
2.5.3 Playtesting	40
2.5.4 Version 2 – Evaluation	40
2.5.5 Workflow Evaluation and Outlook	44
2.6 Version 2 – AI Development	45
2.6.1 AI Architecture	45
2.6.2 Decision Making Analysis	47
2.6.3 Utility AI Design and Implementation	51
2.6.4 AI Configuration	59
2.6.5 Playtesting with Human Players	60
2.6.6 Version 2 – AI Evaluation	61
2.6.7 Workflow Evaluation and Outlook	62
2.7 Version 3 – Competitive Prototype	64
2.7.1 Conceptualization	64
2.7.2 Area Control	65
2.7.3 Economy	70
2.7.4 Turn Counter Mechanics	73
2.7.5 Map Design	76
2.7.6 Gameplay Balancing	80
2.7.7 Visual Development	82
2.7.8 Playtesting	86
2.7.9 Version 3 – Evaluation	88
2.7.10 Workflow Evaluation	95

2.8 Version 3 – AI Concept 97

- 2.8.1 Core Utility System 97
- 2.8.2 Additional AI Systems 98
- 2.8.3 Optimizing for Player Experience 99

2.9 Outlook. 100

- 2.9.1 Possible Milestones 100
- 2.9.2 Gameplay Development. 101
- 2.9.3 Meta Game 102

3. Discussion 103

3.1 Workflow 103

- 3.1.1 Adaptive Game Design 103
- 3.1.2 Adaptive Prototyping 107
- 3.1.3 Adaptive Programming 109
- 3.1.4 Integrated AI Development 110

3.2 Fields of Work 113

- 3.2.1 Game System Design 113
- 3.2.2 AI Development 114
- 3.2.3 Programming 114
- 3.2.4 Working Alone 115
- 3.2.5 Conclusion 116

Figure References 118

References 119

List of Tools 120

Note of Thanks 121

Statement of Authorship 122

1. INTRODUCTION

1.1 Plan and Adjustments

The proposition of my bachelor project was to develop a strategy game with an AI opponent. The game's design would start out simple and be progressively evolved utilizing an iterative workflow. My initial goal was for the AI development to be fully integrated in the game design process in order to ensure optimal quality of both aspects and see if any interesting interactions would emerge. Throughout the project, however, it became apparent that this approach wasn't practical, at least for the particular game I was designing. Instead, I had to switch to a development rhythm that alternated between game and AI design. This led to the situation that I finished my first fully functional AI player relatively late in the project's timeframe. At this point, I decided to prioritize game design over AI development and spent my remaining time on one last, ambitious design pass. While this decision resulted in a final prototype that actually doesn't implement a functional AI opponent, it enabled me to achieve my goal of creating a compelling strategy game that I can be proud of as a game designer.

1.2 Motivation

My main motivation for this project was to work in the two game development fields that interest me the most – game system design and technical design – in order to gain valuable experience and a better understanding of how I want to advance my professional career.

Strategy games are perhaps my favorite genre, yet, throughout my studies, I never had the chance to design one. Therefore, I saw this project as my last opportunity to make a personal game incorporating my favorite game systems, mechanics and dynamics. Furthermore, I was always interested in working on the edge between design and programming. This vaguely defined field, called technical design, encompasses many different areas, one of which is AI development. Game AI is absolutely fascinating to me. I love to design and build automated processes that can be run by the computer to generate a magical world of emergent behavior. To end my bachelor studies with a project about game AI was actually somewhat of a full circle, as I had initially applied to the school with a project about AI agents.

My ambition to develop the AI in parallel to the game design originated from negative experiences in my internship. If the game AI is necessary to play and experience the game, then game and level designers are reliant on its operational state to do their work. If, however, the AI development constantly lacks behind other development processes, it not only leads to diminishing returns on design efforts, but also naturally results in a suboptimal final Game AI at release. Of course, there is an argument to be made that AI development shouldn't follow the game design too closely. If the game system changes – and it probably will – the game AI has to be adjusted and prior work must be discarded. This can be seen as a waste of time and resources, but I had the feeling that this was just a cheap excuse.

Ultimately, I wanted this project to prove my ability to conceptualize and implement both great game design and technical systems and the final prototype was meant to serve as a quality portfolio piece to assist my transition into the professional game industry.

1.3 Scope of Work

Over the course of this project, I created three distinct game versions. The first version was a simple prototype of the most fundamental gameplay ideas, which was realized both on paper and in digital form. The second version extended the digital prototype with a couple of key gameplay mechanics to create what I consider the core loop of the game and can be played against a fully functional AI opponent. Finally, the third version of the game introduced a wide suite of new mechanics to further enhance and refine the game system, resulting in a far more complex and interesting strategic gameplay. This version, however, can only be played by two human players on one computer.

I have decided to hand in two playable prototype builds – one for Version 2 that automatically starts a match against the AI opponent and another for Version 3 that starts a multiplayer game between two human players on my favorite map. Of course, the prototype of the third version can be played by one person controlling both players to test out the gameplay. Because this version doesn't include a dedicated tutorial to teach the gameplay, I created a short explanatory guide. Additional deliveries include digital documentation material, like pictures of various development stages and gameplay videos.

LIST OF DELIVERED WORKPIECES

- Version 1 – 10 Screenshots
- Version 2 – Unity Project
- Version 2 – Prototype Windows Build (playable against AI opponent)
- Version 2 – Playthrough Video
- Version 2 – 15 Screenshots
- Version 3 – Unity Project
- Version 3 – Prototype Windows Build (playable with two human players)
- Version 3 – Gameplay Instructions
- Version 3 – Playthrough Video
- Version 3 – 25 Screenshots

2. DEVELOPMENT

2.1 Development Principles

Before diving straight into development, I made sure to define some key principles for the project that would get me started in the right direction and guide me over the coming months. These principles were informed by my experiences gathered in the three years of studying game design and working on multiple long-term projects.

2.1.1 Simplicity

Simplicity is one of the most fundamental pillars of good design. A product or experience should consist only of its essential parts and every bit of unnecessary clutter can dilute its quality significantly. Despite the omnipresence of this principle, I have struggled with it in practice. Time and time again, I vowed to keep things simple and failed miserably. I often struggled to let go of problematic ideas, tried to fix fundamental problems by adding more elements and at times, I just wanted to impress with sheer quantity. After repeatedly experiencing the negative consequences of useless complexity, as well as seeing others reap the rewards of following the principle of simplicity, I declared once again to try to adhere to it in this project as best as I can.

Naturally, simplicity can also afford practical advantages since a simpler game system will be easier and faster to implement technically and artistically. Because of my limited time and resources for this project and the ambitious goal of developing an AI player, I aimed at a small scope for the game. As “the simplest way to achieve simplicity is through thoughtful reduction” (Maeda, 2006, p. 1), I planned on regularly taking the time to think deeply about trimming and cutting out elements of the game’s design. With this practice of repeated simplification, I hoped to not only develop a great game design, but do so in an efficient manner that would allow me to end the project with a finished playable prototype.

2.1.2 Prioritizing Gameplay Experience

When it comes to game design, there are generally two perspectives – that of the player and that of the game system. While some designers like to focus more on the psychology of play or the interfacing between humans and digital games, I tend to be more passionate about fundamental systems, processes and mechanisms. I love to design, build and configure game systems and sometimes I can be so engrossed in all the models, logic and numbers that I lose sight of the player’s perspective. But, of course, the ultimate goal of the game designer is to create an experience (Schell 2008, p.10). Therefore, I decided to prioritize the gameplay experience from the very beginning and remind myself of this principle throughout the development process. For a strategy game, an enjoyable experience generally comes down to interesting, meaningful decisions and the specific details of the system should always assist this goal.

The same concept had to apply to the development of the AI opponent. While it would be easy to get lost in the complexity of game AI and its numerous techniques and methodologies, in the end, the priority must lie on providing a gratifying gameplay experience for the human player. As game AI is an immensely difficult task and bad AI can easily ruin a game, I set my ambitions relatively low. I wanted to develop a simple AI system that was just sophisticated enough to get the job done without being annoyingly dumb.

2.1.3 Iterative Workflow

Perhaps the most crucial learning I made throughout my game design studies is the importance of an iterative workflow. It is basically impossible to create a game concept that just works out perfectly because systems, consisting of multiple active, interconnected elements, will inevitably create unforeseeable emergent dynamics (Adams & Dormans, 2012, p. 23–58). An effective workflow must be able to adapt to this unpredictable design space by utilizing iterative cycles.

The iterative design process has a long history and comes in numerous slightly varying forms in the many fields of design and engineering (Larman & Basili, 2003). In game design, the process is most often formalized into four distinct steps that Macklin and Sharp (2016, p. 106–116) define as conceptualization, prototyping, playtesting and evaluation (**Figure 1**). In practice, game designers conceptualize an idea and build a prototype which is playtested. The playtest is evaluated and the findings are fed back into the next cycle to improve the concept.

Many times have I experienced what happens when game development lacks iteration. In the beginning, a complex design for the final game is conceptualized. This is followed by a drawn out prototyping phase that takes up almost the entire development timeframe and when the game finally comes together to be played for the first time, it is revealed to be flawed in many ways. But with no time left to implement meaningful changes, the final result remains suboptimal and disappointing.

To avoid this pitfall, I planned on implementing a high frequency of iterations. In my experience, the most critical step of an effective cycle is the playtesting, because it's the valuable information generated in actual interactive play sessions that really drives the feedback loop of iterative design. Therefore, I intended to focus heavily on the constant playability of my prototype and planned on enforcing a regular schedule for playtests.

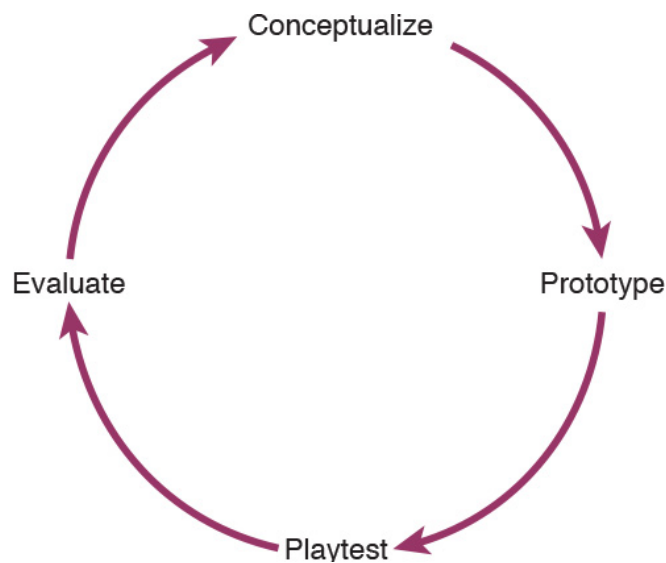


FIGURE 1
Iterative design
process

2.1.4 Clear Vision

Another decisive requirement for the successful development of a game is a clear vision. While it is possible to generate a great game concept with spontaneous brainstorming and experimental prototyping, this is by no means guaranteed. For a short, low-risk project, like a game jam, this can be very fun and educational, but for a bigger project it is a gamble that can turn out disastrous. Therefore, it was important to me to begin this project with a clear vision of what the fundamental gameplay would look like. To achieve this, I decided to utilize well established mechanics of the strategy game genre and carry out an extensive early ideation phase (which is discussed in section 2.2). Of course, game design is unpredictable and it is crucial to stay flexible and adaptive in the development process, but in order to achieve my goal of iteratively evolving the game design and creating an AI player, I needed to avoid wasting time on prototyping an ill-defined vision.

2.2 Ideation

I began the game design process with an extensive ideation. On one hand, as discussed earlier, I wanted to define a clear vision for the game, on the other hand, however, an overly concrete concept would stand somewhat contradictory to the adaptive game design approach. It didn't make sense to design the final specifications of a game that was supposed to undergo an evolutionary design process. Instead, I wanted to create a collection of ideas for gameplay mechanics which are somewhat interconnected but ultimately independent and modular. This way, I hoped, I could start out prototyping with a few of the modules and then add others in following design iterations as needed.

2.2.1 Ideation Guidelines

Before going into the actual ideation and collection of gameplay ideas, I defined two key guidelines for the process.

The first guideline was that I wanted to draw inspiration from my own gameplay interests. It's not necessarily a given that a game designer can work on a game that overlaps with his own interests, but as this was my first solo project in a long time, I decided to delve into the strategy and management genre and work with my favorite mechanics. Besides being more passionate and driven about the project in general, working with familiar game systems would also allow me to rely on a lot of gameplay experience. Furthermore, utilizing established, proven mechanics would reduce the risk of getting stuck early in development.

The second guideline related to the development of the AI player. AI development for games can be very difficult and as my experience in the field is relatively limited, it was crucial to keep the challenge manageable. Some gameplay mechanics are just inherently more challenging for an AI and, of course, as a strategy game gets more complex, an artificial player requires a more sophisticated decision making algorithm. Consequently, I planned to evaluate all my ideas from the perspective of how difficult it would be to make the game AI handle them.

2.2.2 Buildings and Area Control

Buildings

The core mechanic of the game would be the placement of buildings. Placing interactive elements on some form of environment is a well established and proven gameplay action that forms the cornerstone of many strategy and management games. While playing *Islanders* (Figure 2), I realized how powerful this mechanic can be, because placing buildings is basically the only action players can take in the game and it is both interesting and challenging for many hours. The strategic depth comes solely from the

interconnected relationships between buildings and the environment. This stand-alone quality of the building placement mechanic convinced me that it was the perfect core piece to build a simple yet compelling game. In addition, I was certain that the task of placing elements on a map would be easily achievable for the Game AI. An algorithm that computes a score for all possible positions and picks the best should be relatively easy to design given clearly defined rules for the attractiveness of locations.



FIGURE 2
Building placement
in Islanders



FIGURE 3
Castle exerting
implicit map
control in
Age of Empires 2

FIGURE 4
Distinct provinces
in Europa
Universalis 4



FIGURE 5
Explicit territory
control in Settlers 3



Area Control

The second key element of gameplay was going to be area control, which is an essential dynamic of many games. Battling for territory provides an interesting challenge and seems to trigger a deep psychological desire for power and control. In some games area control emerges as a dynamic of interactive game objects in the playspace. In chess, a well-placed piece can control the board space around it and in a real-time strategy game like Age of Empires 2 (Figure 3) a player might control their flank by placing a

strong military building that can repel incoming attacks. In other games, area control is explicitly built into the mechanics of the system. In the classic board game Risk and modern digital variations like Europa Universalis 4 (Figure 4), the world is compartmentalized into regions or provinces which are literally controlled by players. Yet another form of area control can be found in games where game objects exert explicit control over space. One example is Settlers 3, one of my all-time favorite games, in which players build military buildings that take control of the surrounding area (Figure 5). Other buildings can then be freely placed inside the controlled territory and by making use of the resources inside this space, players can afford to build more military buildings. This dynamic creates a gameplay loop of expansion and economy management, which I was hoping to recreate by making the placement of area control buildings the starting point of my gameplay exploration.

Combinatorics

Another major gameplay idea was the spacial combinatorics of buildings. There are some games, like the Anno series (Figure 6), in which placeable elements have to be rotated and fit together to make efficient use of limited space. This synergizes well with the area control dynamic, but generally the shapes of buildings in these games are fairly simple and, hence, the combinatorics are relatively trivial. I saw a lot of potential to expand upon this idea and envisioned more elaborate shapes, which would provide a trickier combinatorial challenge for players, similar to a puzzle. Of course, such a system would also be demanding for the AI player, but I was sure this could be handled with some form of look-ahead technique like the minimax algorithm (Fan, 1953).



FIGURE 6
Optimization of
building space in
Anno 1404

2.2.3 Resources and Economy

Resources

The next group of ideas revolved around developing some form of economy. I planned on having basic resources physically distributed on the map, which would be gathered by specific buildings. I wanted these resource deposits to be depletable to fit well into the area expansion dynamic and force players to constantly spread out into new territories. The finite nature of resources would also hopefully help to limit the possible scope of the game.

Production

While basic resources form the backbone of any economy system, production mechanisms will extend its complexity and afford more interesting challenges to players. The management and careful configuration of complex production chains is at the heart of many of my favorite games. Some, like Anno or Factorio, take production chains to the extreme, but I wanted to keep them relatively simple and examine them in the context of the area control dynamic. Simplicity would also ensure that I would be able to develop a Game AI that could perform well in the economic system.

Auction Trading

Another idea of mine was auction trading. Of course, trade is a fundamental element of economics and I was very interested in it, because trading with opponents would be a great way to have active interaction between human players and the AI. But direct trading is complicated and can become frustrating when players have to spend a lot of time configuring trade agreements. Some games get downright annoying by spamming players with trade requests from AI factions.

Because of this I conceived a well-structured auction trade mechanic. Periodically the gameplay would be interrupted by an auction in which all players engage in a bidding contest for a fixed amount of resources. Potentially they could also bid on selling resources. I envisioned some form of trade ship that sells off its cargo and then buys up new resources to take away. Human and artificial players would only have to decide on how and not when to trade. This mechanism, I thought, would not only create a better rhythm for the overall gameplay, but also be comparatively easy to handle for the Game AI by some form of utility function which estimates the value of resources.

2.2.4 Building Cycle

In the beginning I targeted a playtime of around 10 to 20 minutes. This served as an additional constraint on the possible complexity of the game system and should ensure that playtesters could experience the full gameplay arc in a reasonable timeframe. To achieve this, I wanted to avoid a game in which players build up a complex economic system indefinitely and came up with the following gameplay mechanisms to create a short and dynamic gameplay experience.

Dismantling Buildings

A strong focus of the game would lie on dismantling buildings. Once the finite resources of a region would be depleted, obsolete gathering buildings would have to be dismantled and a small resource refund could be used to subsidize further progression. Additionally, at the end of the game, all players would have to abandon the game world and leave everything behind, except their resource inventory. This would result in a special late game phase in which many buildings have to be dismantled to get back the most valuable assets.

I felt like this mechanic was one of my more original ideas, since not many games focus on dismantling as a core mechanic of their gameplay loop. But while I saw great potential for a unique selling point, I also worried about potential problems. There was a good chance players would be reluctant to disassemble their creations. I would have to find a way to manage their expectations and make them comfortable with the concept of impermanence.

Interdependence

Another quite obvious gameplay idea was the interdependence of buildings. Some buildings would have effects on others. This could come in the simple form of bonuses like increased gathering speed or production rates, but also as real dependencies where a building can only be placed next to some specific other building. I wasn't sure yet how these effects and dependencies would work in detail, but I was confident that this would enhance both the combinatorial puzzling and dismantling dynamics and make decision making more interesting.

Maintenance

One last important concept for buildings was maintenance. Maintenance would be a continual cost for existing buildings, creating a negative feedback loop which was supposed to limit growth and expansion. As players would obtain more control and buildings, the rising maintenance cost would slow them down and, in theory, worse off players should be given a chance to catch up. In addition, the cost of any existing building would further incentivize the use of the dismantling mechanic.

2.2.5 Game Structure

In some games, like chess, distinct gameplay phases emerge from simple rules and mechanics alone. Other games utilize more specific methods to structure the progression of gameplay. At this stage, I wasn't sure which type of game I was making, but I thought it would be good to develop a few ideas.

Starting Placement

The first idea was an elaborate starting placement phase. This was somewhat inspired by the classic board game Catan (Figure 7) in which players take turns to place their first settlements on the map. This crucial action has great impact on the rest of the game and is one of the most important and interesting strategic decisions players have to make. I envisioned the starting placement phase of my game to be some form of auction, similar to the trade mechanic.

FIGURE 7
Starting placement
influencing the
progression of the
game in Catan



Milestones

I wanted to include some form of milestones throughout the gameplay arc, which were meant to be momentary or permanent goals for players to work towards competitively. Players would be able to make interesting decisions about which milestones to compete for, like in Catan, where players have the option to go for different milestones, like the longest road or the largest army, which grant additional victory points. For my game I envisioned special buildings that can only be built once and give players special rewards tied to the different gameplay systems.

End Event

To create a thrilling end game, I planned on having some form of special end event that concludes the game. I thought some form of imminent ultimatum would enhance the late game phase in which resources are depleted and players must dismantle their last buildings and leave the game world. To make this even more interesting, players could also have the ability to influence this event in some form.

2.2.6 Design Pillars

After working through so many specific ideas, I felt like I was getting lost in specifications. Therefore, I decided to conclude the ideation phase by defining a few abstract game pillars that would apply no matter what the specific mechanics would look like. By evaluating the game design through the lense of these pillars regularly, I wanted to make sure to stay in line with my original vision.

Interesting Decisions

For the first game design pillar I chose interesting decisions. For a strategy game to be interesting, it needs to provide players with multiple viable options in any given situation. While this might seem obvious, it can be surprisingly difficult to achieve in practice. As mentioned before, in the heat of development, I tend to get bogged down in the details of the game system and sometimes lose track of what the player is actually tasked to do while interacting with it. Hence, I planned to constantly remind myself that interesting decisions are the absolute essence of what makes a strategy game tick and make them the first priority of my design efforts.

Multiple Strategic Paths

The second pillar – multiple strategic paths – relates directly to the first pillar, but goes a little bit further. In order to afford an interesting decision, a game only needs to present players with immediate options that might have little impact on the overall game. Meanwhile, decisions about strategic paths should have long-term implications for the whole gameplay arc. They represent different ways to approach the gameplay fundamentally and I wanted my game to include at least a couple of them.

Competitive Interaction

As the third game design pillar I declared competitive interaction between the players. Some multiplayer games, like many euro-style board games, tend to lack in this department. For example, in the complex strategy game *Terraforming Mars* (Figure 8) players act on the same playspace and influence the same systems, but actual direct interaction between them is very rare. There are no mechanics with which players can actually affect their opponents' play in a meaningful way and in the end, it feels like everyone is playing their own game at the same table. I wanted to make sure that my game provides players with actionable tools to really interact with their opponents. This, of course, would also mean direct interaction between human and AI players.

FIGURE 8
Players control tiles of a shared playspace in *Terraforming Mars*



Open Information

Last but not least, the fourth pillar, I decided, was the concept that is known as open information in economics and game theory (Gibbons, 1992). This meant that the entire internal state of the game would be available to players. My decision to design an open information game was mainly made to provide a system in which the Game AI can completely mirror human players and make its decisions by action on precisely the same information. This would enable the romantic idea of creating an AI player that simulates a human being without any additional tricks or cheats. Additionally, it was also meant to prevent the game design from getting too complex by adding any complex underlying mechanisms that aren't comprehensible to human players.

2.2.7 Narrative Concept

While doing my extensive ideation, I also came up with some form of narrative concept for the game. Generally, however, I tried to be very careful with this. While a narrative concept can be a fruitful basis for many gameplay ideas, in my experience, it can also severely limit design thinking and flexibility. The mind starts to oppose potentially interesting game mechanics because they seem to contradict the narrative. Therefore, I tried to apply the earlier described philosophy of undetermined flexibility to narrative ideas as well.

The setting I always came back to was that of a distant planet. The surface of the planet is the playspace and players are companies that arrive with the help of some sort of powerful megacorporation in order to exploit the planet's resources. This set-up would provide a context in which players clash in a semi-regulated competition, which revolves around claiming land without any explicit warlike actions. The narrative also fit perfectly with my planned gameplay phases – arrival, build-up and ultimate departure – and would hopefully help to legitimize the dismantling mechanic.

2.3 Fundamental Technical Systems

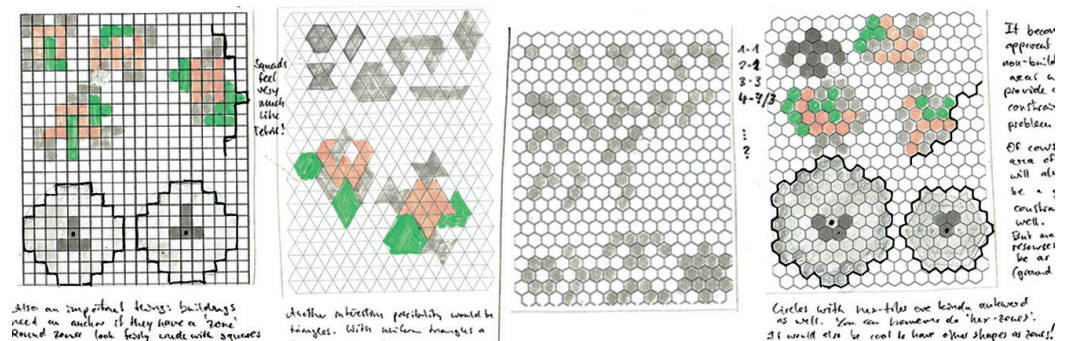
After the ideation phase, I started to implement the most fundamental technical systems which would be required to start the prototyping phase. All of these systems would have to be extended and adjusted throughout the whole development process, but the ground work and core functionalities were created at this early stage. I generally want the focus of this paper to lie on the design aspects of the project and not go too deep into programming. Therefore, this section will be no more than a brief overview of the most essential technical systems without delving into too much detail.

2.3.1 Hexagonal Map System

Hexagon

Very early in the ideation phase, I decided to use a hexagonal tile map, because hexagons proved to be superior to other shapes, like squares and triangles, in many aspects (Figure 9). I knew that buildings would be composed of multiple tiles and square tile shapes looked very rigid and boring, while triangular shapes had harsh narrow angles which didn't resemble buildings at all. Hexagonal compound shapes, on the other hand, looked interesting and balanced. The same was true for larger areas like the circular ranges I wanted to have for the area control effect. Furthermore, the six sides of the hexagon provided more puzzling possibilities than the four sides of the square because hexagonal shapes can be rotated in more angles and fit together in more combinations.

FIGURE 9
Experiments
with different
shapes for grids,
buildings and areas



Hexagonal Grid System

Hexagonal grid systems are by no means trivial. However, as they are a classic component of many digital games, it's not difficult to find quality information on the topic online. I read through the great blog posts by Red Blob Games (Patel, 2013) to get familiar with the particular mathematics of hexagonal grid systems and used the information as the basis of my own technical implementation.

The core piece of my hex map system (**Figure 10**) was the Hex data structure, which basically described a three dimensional coordinate (**Figure 11**) and provided various helper functionality for arithmetics, rotation, neighbor access and many more. Built on top of this were other data structures like the Hex Edge to modulate borders. I thought about creating a dedicated Hex Area class for groups of hexagons, but ultimately just used the generic C# list data type and a static Hex Geometry class for functionality such as generating, transforming and modulating areas.

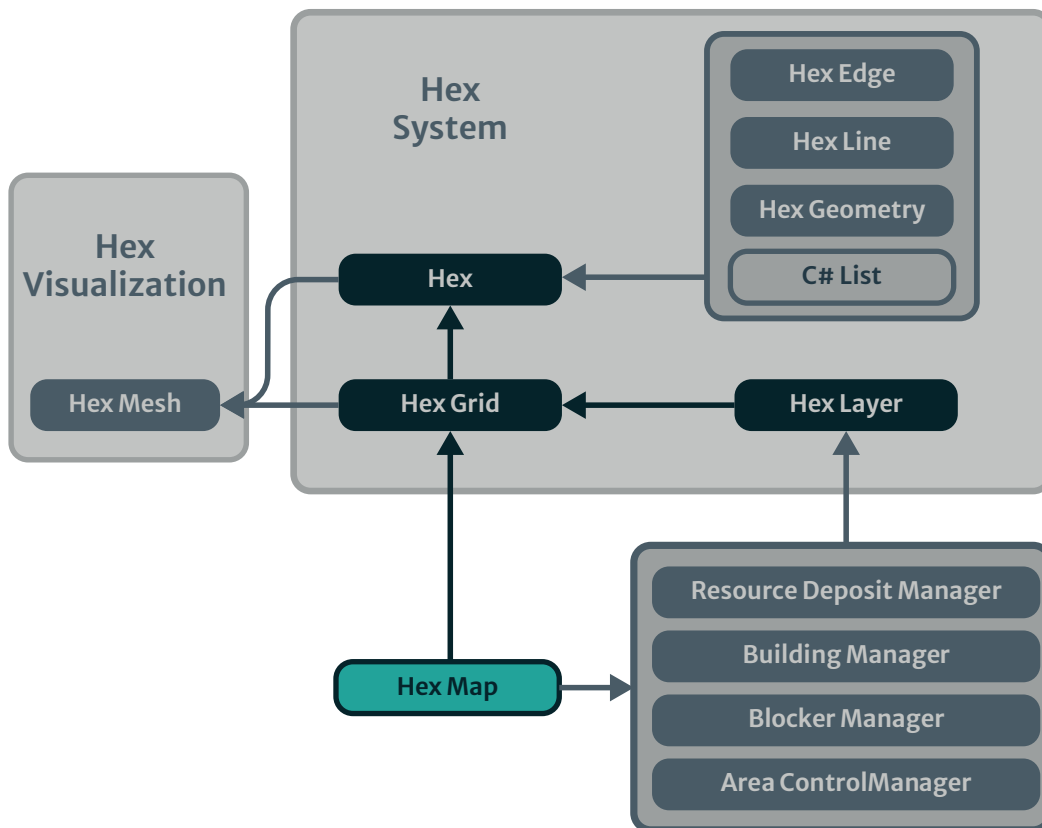


FIGURE 10
Hex Map System
Architecture

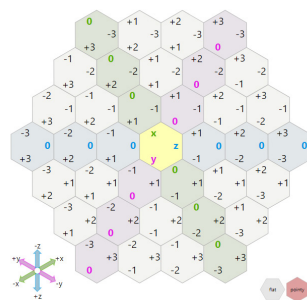


FIGURE 11
3-dimensional
modulation of
hexagonal
tile grid

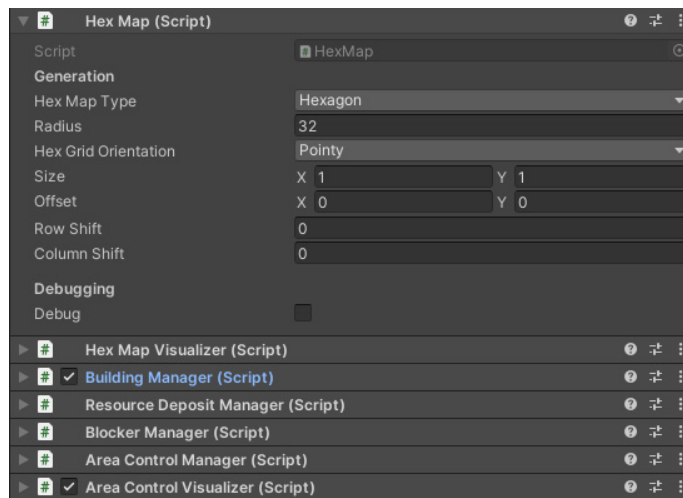
The basic compound data structure to combine multiple hexes was the Hex Grid class. This class held a 2-dimensional array of Hex types and spacial information like orientation and scale, as well as extensive methods to convert from hex space to world space and back.

The last data structure of my core hex system was a generic Hex Layer class. A Hex Layer could store any data type inside the dimensions of the referenced Hex Grid. This was used for any type of information that needed to be put inside the grid structure, like map entities, area control state and AI information, such as influence maps.

Hex Map System

In order to tie everything together I created a Hex Map singleton class. While the singleton pattern is a somewhat controversial system architecture solution (Nystrom 2014, p. 73), it enabled me to create an easy global entry point to access all data concerning the game map and it's contents. The Hex Map class referenced multiple manager classes, which implemented the Hex Layer data structure, to organize buildings, resource deposits, area control and more (Figure 12).

FIGURE 12
Hex Map and
manager classes in
Unity Inspector



An abstract base class called Map Entity was the basis for all concrete objects that could exist on the map and held fundamental information like the anchor position. It was extended by classes like Building and Resource Deposit which added their own specialized data and functionalities.

Area control was modeled by a Hex Layer which held instances of the Hex Control class. Area control buildings could lay claim to specific hexes and the Hex Control class would then calculate which control state it would take. This area control algorithm was still very simple at this early stage, but should become one of the most complex and challenging technical tasks of this project, which is discussed in section 2.7.2.

Hexagon Visualization

Another important part of the Hex Map was, of course, its visualisation. The core of my hexagon presentation system was the Hex Mesh class which created a single hexagonal mesh given a hex coordinate and the spacial information of the Hex Grid. Further functionality allowed for combining single meshes into compound shapes and creating linear meshes to visualize borders.

2.3.2 Generic Player Architecture

It was clear from the beginning that this game would have multiple participating factions, which would function exactly the same in gameplay. The control of these factions, however, would differ between human and computer-controlled players, which required a player architecture that separated the core functionalities from its control modules (Figure 13).

To stay in line with the narrative of my game, I called a faction a company. The Company class contained some basic identification information and referenced owned buildings and an inventory to store resources. It was able to act in the game world via specific executive functions, which could be called by a company controller. In the end, a company didn't care how it was controlled, which made the system modular and easily extendable.

The Company Controller class was an abstract base class which took ownership of a company. The two concrete implementations of this were the (human) Player and the AI Player classes. While the human player controls the company's executive functions with certain interface functionalities, the AI player does so via automated decision making algorithms.

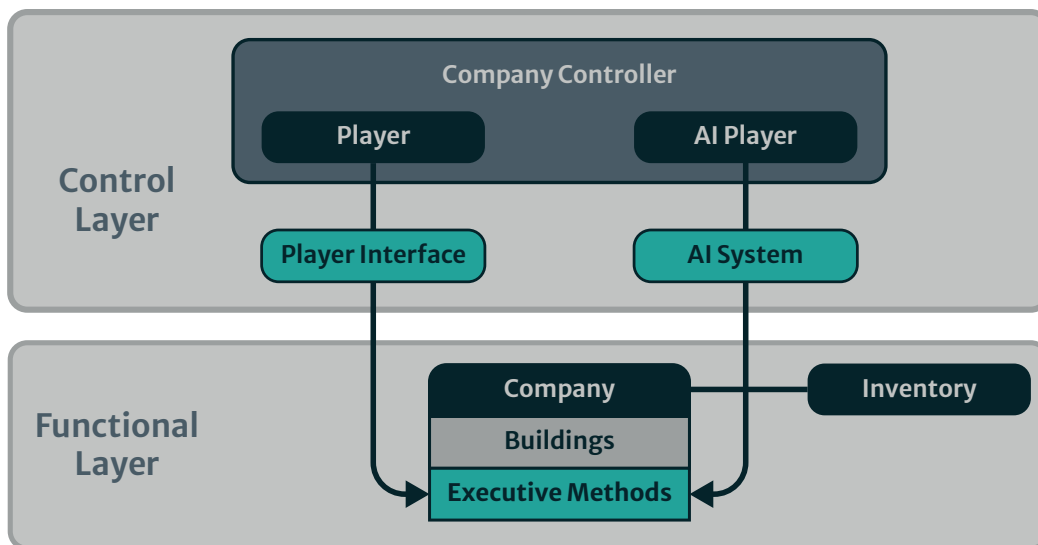


FIGURE 13
Generic Player
Architecture

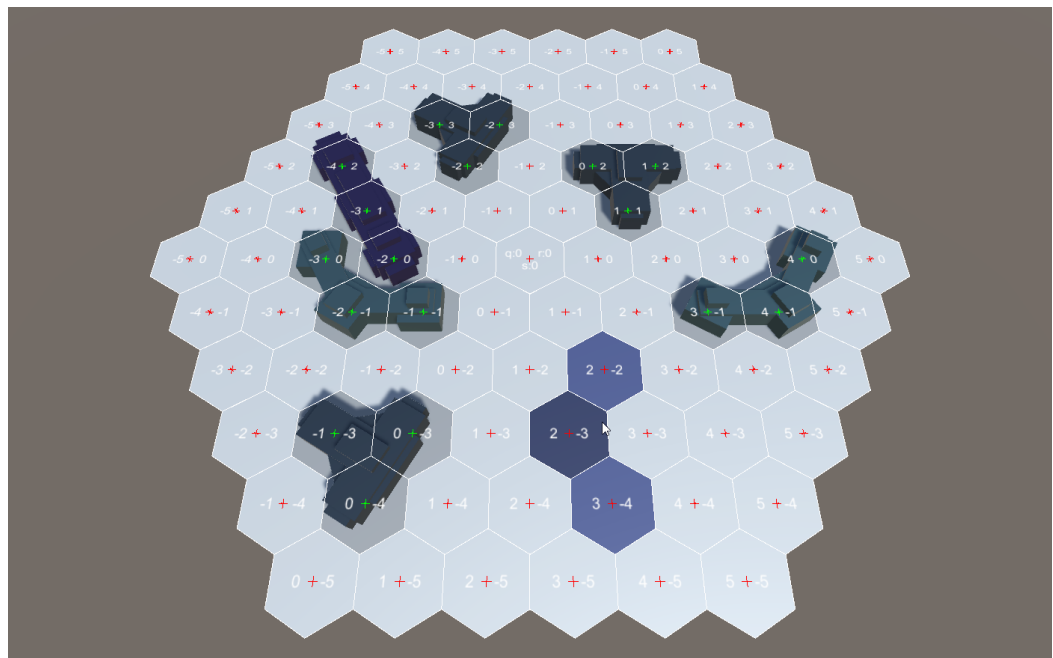
2.3.3 Human Player Interfaces

Mouse Controller

As stated before, this was the first time I developed a serious strategy game. In the many projects throughout my studies, I mainly worked on games where the player controls a physical player character, which necessitated the technical implementation of a character controller, involving complex physical algorithms, responsive controls and the handling of many troublesome movement edge cases. So going into this project, I thought that I would have an easier time without the need for a character controller, but as it turned out, I had naively underestimated the technical requirements for what I would call the Mouse Controller.

The player would use the computer mouse to interact with the game world by clicking, dragging, scrolling and so on. A simple screen to world raycast determined the position of the mouse pointer on the map and could detect interactable objects. Mouse movement had to be tracked even when no clicking was happening in order to register entering and exiting world objects for possible hovering functionalities. Some entities were selectable by clicking and the selection state could be exited by further clicks. Clicking the building buttons in the graphical user interface (GUI) would enable the placement mode. Then, when hovering over the map, a ghost building was visualized to represent the placeable building. If it couldn't be placed at the current position, it had to be visually changed to indicate the impossibility of placement. At last, all the hovering, selection and placement visualizations have to be either retained or cleaned up in all sorts of specific situations.

FIGURE 14
Simple prototype
for building place-
ment on a tiled map



In the end, even though I was quite surprised by how complex the logic and visual communication for this basic player interaction had to be, I managed to successfully create a simple prototype in which the player could place simple entities on the tile map (Figure 14).

Camera

While developing the different interfaces, I analyzed many existing strategy games and drew heavy inspiration from how they handle certain functions and controls. It was interesting to see that most games follow a standard, but then tend to deviate in some aspects to accommodate for certain special requirements of their gameplay. For example, most games allow players to zoom the camera with the mouse wheel, but some disable this functionality in the building placement mode because the mouse wheel is also used to rotate buildings. In my game, however, players still wanted to use the camera zoom when in placement mode, so I had to find another solution. Ultimately, I decided to use the Q and E keys for building rotation, which worked well in conjunction with WASD camera panning.

Another question was whether the camera should be able to rotate in order to look at the game world from different angles. In some games this function is absolutely necessary for players to see all objects adequately. For example, in Frostpunk (Figure 15) the players build a circular base around a central building which blocks the view. In my game, however, different rotational views actually disoriented a lot of players because the strict hexagonal grid is best viewed from a perfect angle. Therefore, even though I personally liked to view the game from different angles, I decided to cut camera rotation. Later in development, this would actually have quite an impact on what building shapes were acceptable because any extravagant shapes that blocked the players' view would have to be avoided.

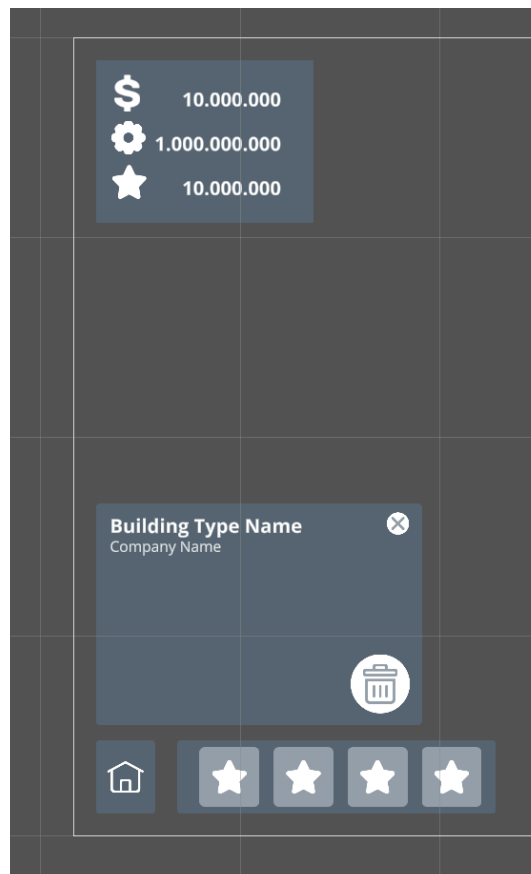


FIGURE 15
The central generator blocks the view of players in Frostpunk

Graphical User Interface

Strategy games depend heavily on a good GUI, as the game must communicate a lot of information to players. Many abstract actions can't be represented in the game world and have to be expressed by clickable buttons. Of course, the focus of this project was the game design and not any fancy visual development, so I decided to use a modern flat graphical style. I knew that the GUI would have to adapt to the coming game system changes, but I tried to already establish a reasonable layout, which, at this stage, included a simple inventory panel, a building selection bar and a window for displaying information about selected buildings (Figure 16).

FIGURE 16
First version of a
simple GUI



2.4 Version 1 – Minimum Viable Prototype

Once the fundamental technical systems were in place, I was eager to create the first playable prototype of the game. As discussed earlier, I wanted to start with a simple version and then carefully add complexity in consecutive iterations. Therefore, I wanted this first iteration of the game to be as elementary as possible.

2.4.1 Minimal Economy

So the big question was what the first iteration would look like. What would be a good scope for this minimum viable prototype? Which gameplay mechanisms must be included because the game system can't possibly work without them and how many resources and buildings are needed to cover them?

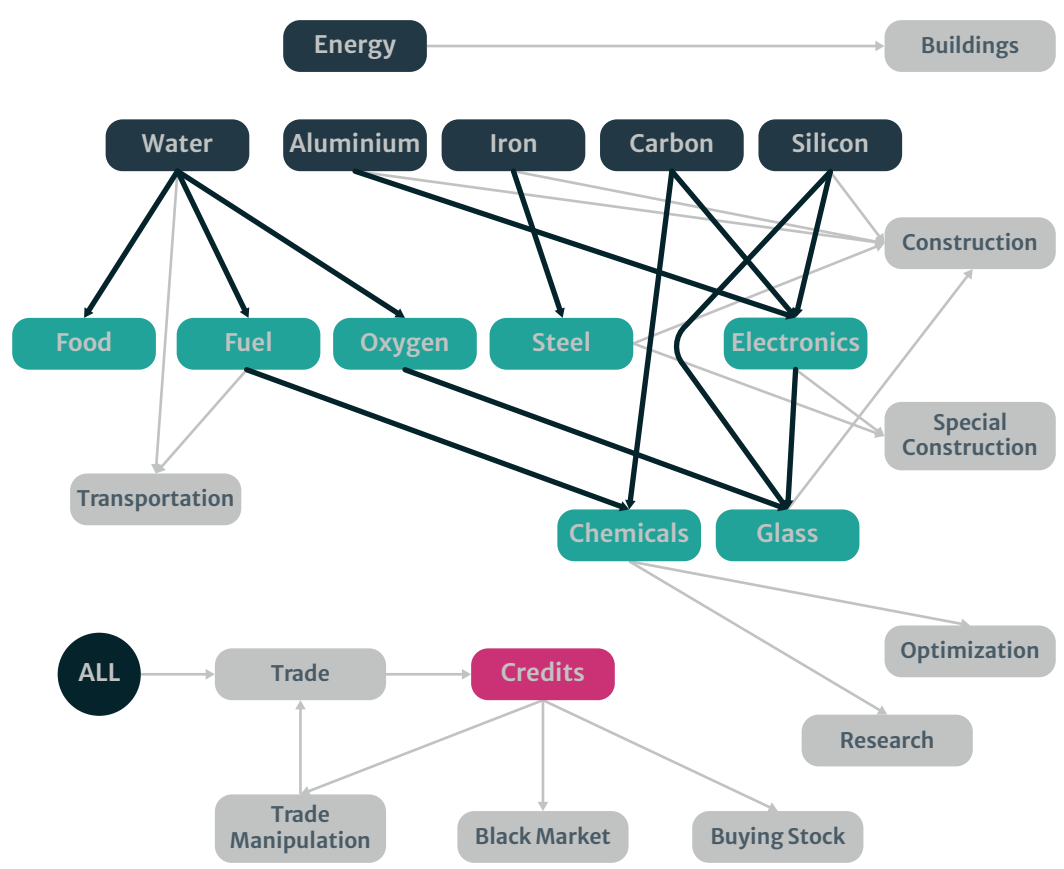
At this stage I analyzed multiple games that are similar to what I wanted to create. One commercial game that comes very close to my vision is *Offworld Trading Company* (**Figure 17**). Especially the narrative overlaps strongly. It is a real-time strategy game with a heavy focus on economy which works without any true military gameplay. The players control companies and are tasked to exploit the planet Mars for profit. They gather and produce resources and trade them on a dynamic real-time marketplace. The players that make the most money can buy-out shares of opposing companies and take them over to win the game. Some crucial differences to my concept are that companies do not really control areas and resource deposits are not finite. This means the focus of gameplay in *Offworld Trading Company* is not so much on spacial territory disputes, but on production chains and trade dynamics.



FIGURE 17
Offworld Trading
Company

In my quest for the essence of my game, I did a detailed analysis of Offworld Trading Company to understand the fundamental structure of an economic gameplay system so I could reduce it to its minimum. The first key elements are, of course, the resources. The game has 14 unique resources, each with distinct roles. The most universal resource is money, which can be exchanged for everything else in the marketplace. It can also go negative in certain situations, as players accumulate debt and must pay interest. The second special resource is energy, which is paid as maintenance for almost all buildings. When analyzing the complex production chain of Offworld Trading Company (Figure 18), it made sense to distinguish between gatherable resources and production resources. While the first can be gathered directly from the map, the second has to be produced from other resources. Two resources can only be produced from other production resources, which results in a production pyramid of 13 resources and 3 layers.

FIGURE 18
Offworld Trading Company's resource roles and relationships



I also looked at other games with production chains, such as Anno 1800, which has a staggering amount of resources, but structures them into separate smaller production chains (Figure 19). This results in many different production pyramids that never

consist of more than 11 resources and 4 layers. A lot of different resources can give depth and strategic options to an economy game, but in order to stay within a reasonable scope, I was sure I could only work with one simple production pyramid. However, I wasn't sure what the lowest number of resources would be without being trivial and boring.

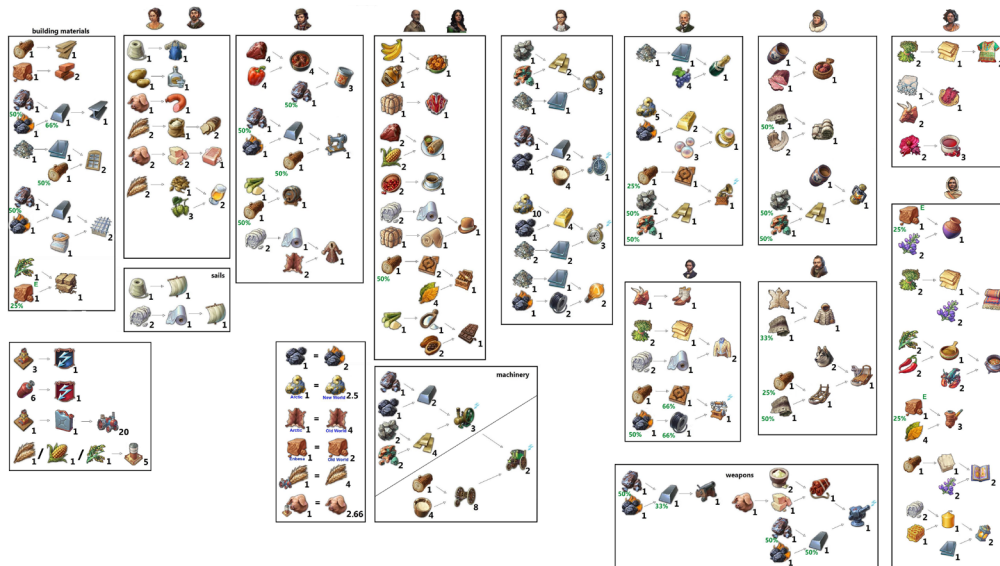


FIGURE 19
Production chains
of Anno 1800

Another important perspective is to look at what mechanics revolve around the individual resource types. The game system of Offworld Trading Company provides not only the basic mechanisms of gathering, production and construction, but also maintenance, trading, research, optimization and buying stock (Figure 20). A lot of the extra mechanisms are enabled by constructing special buildings, which intertwines the different

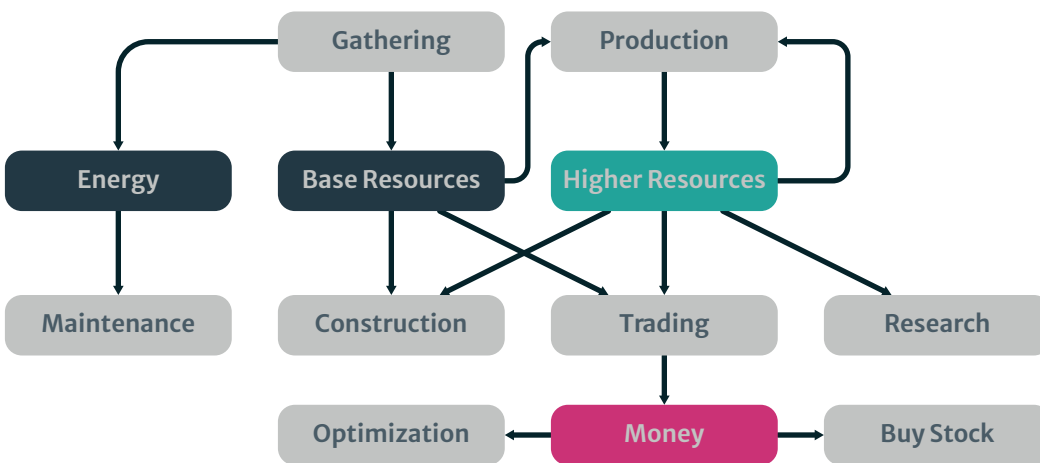


FIGURE 20
Offworld
Trading
Company's
interconnected
game system

gameplay systems even further. One approach to stay within a limited scope might be to reduce the individual parts of the game system but have multiple interconnections between them to create the complexity needed for an interesting strategy game. In my experience, however, interconnections are just as hard to design and develop as the concrete parts of a game system.

In my mind, I needed at least 6 resource types and 3 layers to create an interesting production pyramid that could form a strong core for interesting economic gameplay. When I thought about all the buildings I would need to accommodate for this amount of resources, I realized how large such a system would become. I recognized that the simplest possible way to make my game was already far too complex. But what if I had just looked at too many complex commercial games? Maybe I was already looking too far ahead in the future? Maybe my design method was far too theoretical? This is when I had a sudden epiphany – I needed to drastically shift my perspective and change the design approach completely.

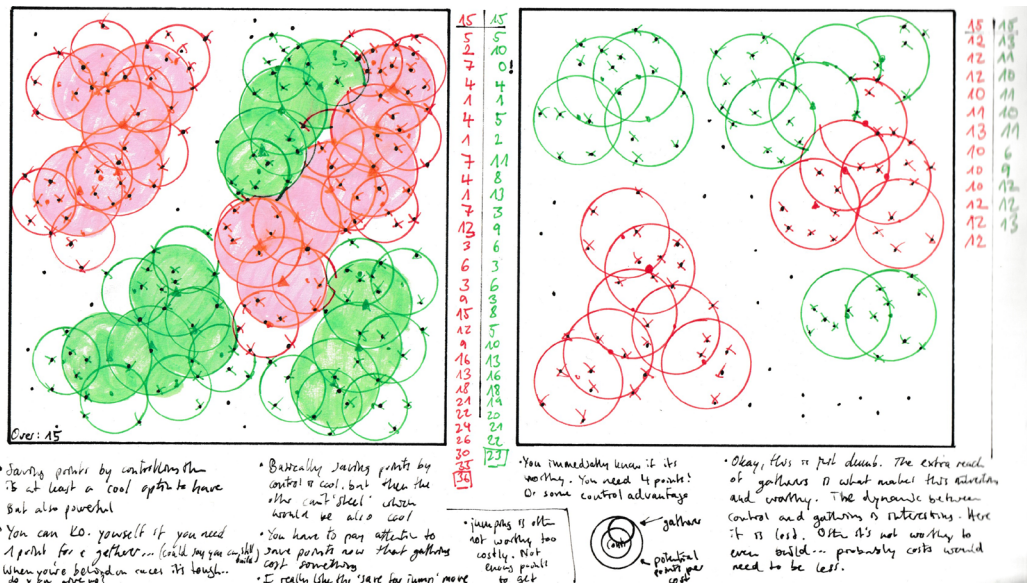
2.4.2 Paper Prototype

In a rush of creative energy I grabbed my notebook and started working on a paper prototype. I approached the minimum viable game question once again, but this time with a practical rapid prototyping method in order to find the absolute simplest way I can make a game about placing buildings, area control and resource gathering.

Simplest Game Possible

Two players would take turns to place buildings on a rectangular playspace which was filled with a number of dots, representing resource deposits (Figure 21). Players would

FIGURE 21
Paper Prototyping



start with a certain amount of resources, which they spend on placing buildings. When placing a building, players would put a simple dot on the map and use a circle template to draw the building's range. The building range of area control buildings was then filled with the color of the player to represent controlled territory. Placing an area control building in an uncontrolled part of the map cost more resources than placing it inside one's controlled area. This made jumping into open spaces more expensive than growing one's existing territory. The gather building could only be placed inside controlled area and would gather resources from all deposits inside it's range. The resource deposits could only be gathered from once and were crossed off afterwards. The game ended when most resources on the map were gathered and both players didn't want to make any additional moves. The player with more resources won.

Game Loop

The most important part of this prototype was it's playability. Interestingly, the simple turn-based mechanics already created an engaging gameplay flow. The two types of actions – taking area control and gathering resources – resulted in a strong game loop (**Figure 22**). Players expanded their territory to gather new resources, which could be reinvested into more expansion. In my search for the simplest game possible, I also tried to play the game with just one building, but the resulting gameplay was far less interesting, as it didn't create a real game loop and lacked the interdependence and tension between expansion and gathering.

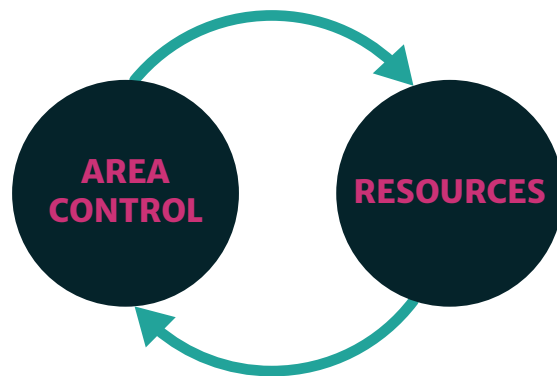


FIGURE 22
Game Loop

Design Iteration

I was surprised by how large the design space of such a simple game system already was. It's few elements provided plenty of opportunity for configuration and balancing. Throughout half a dozen quick design passes, I adjusted the parameters and mechanics of the game and quickly improved the quality of play. It was very interesting to examine the changes that specific adjustments would invoke. Parameters, like the amount of

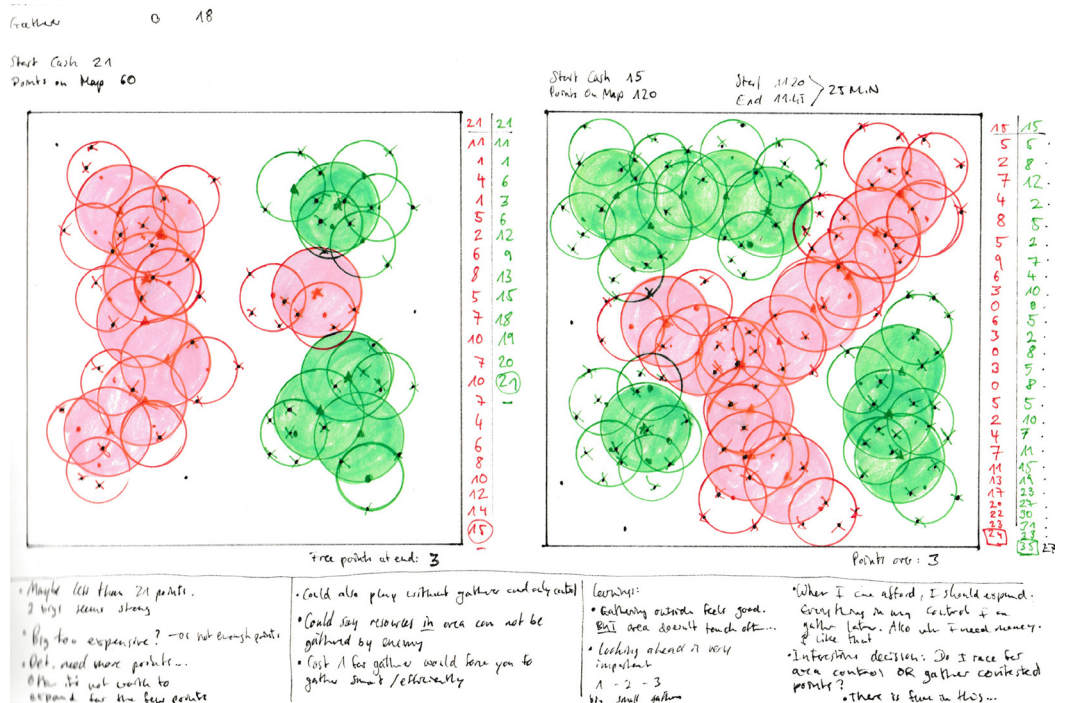
resource deposits on the map, starting funds for the players and building costs, all had a significant influence on how gameplay would unfold. My goal was to enable a balanced flow and end up with values that are easy to understand and memorize.

Furthermore, some gameplay situations emerged that called for more mechanics and rules to be defined. One example was whether resource deposits inside the control of one player could be gathered by another. When this was allowed, it was very fun to steal resources out of the area of the opponent, but when it was forbidden, it was strategically interesting to take control of resource deposits and gather them later. While some design decisions were obvious and easy, situations like this didn't have a clear answer. In the end, it came down to individual taste and the gameplay experience that would fit the overall vision. At this time, I picked the strategic value of controlling resource deposits over the fun of stealing them.

Playtesting

The turn-based gameplay allowed me to play against myself and quickly test out adjustments, which enabled a fast iterative design process that yielded great results. When I ultimately tested the game with other human players, the system was already very polished and the gameplay worked out as intended. With two players the gameplay experience was very competitive and the opponents took a lot of time to think about the optimal actions to take. Even though the game was still incredibly simple and limited, all playtesters were positive that the game was already quite fun and that the core gameplay had a lot of potential to be expanded upon.

FIGURE 23
Paper Prototyping



2.4.3 Prototype Digitalization

I was very happy with how the paper prototype turned out and it was clear that I had established a fundamental game system to expand on. But the paper medium had already reached its limits, as it was quite cumbersome to handle the circle template, fill out controlled areas by hand and calculate all the resource changes. Therefore, I decided to recreate the game digitally and continue on from there. Because I had developed the fundamental technical systems already, this was an easy transition, but some changes and extensions had to be made to the game logic and I improved the area control visualization with area and border meshes (Figure 24). In the end, the gameplay of the digital version felt significantly smoother because a lot of the manual labor and rule maintenance was taken over by the computer.

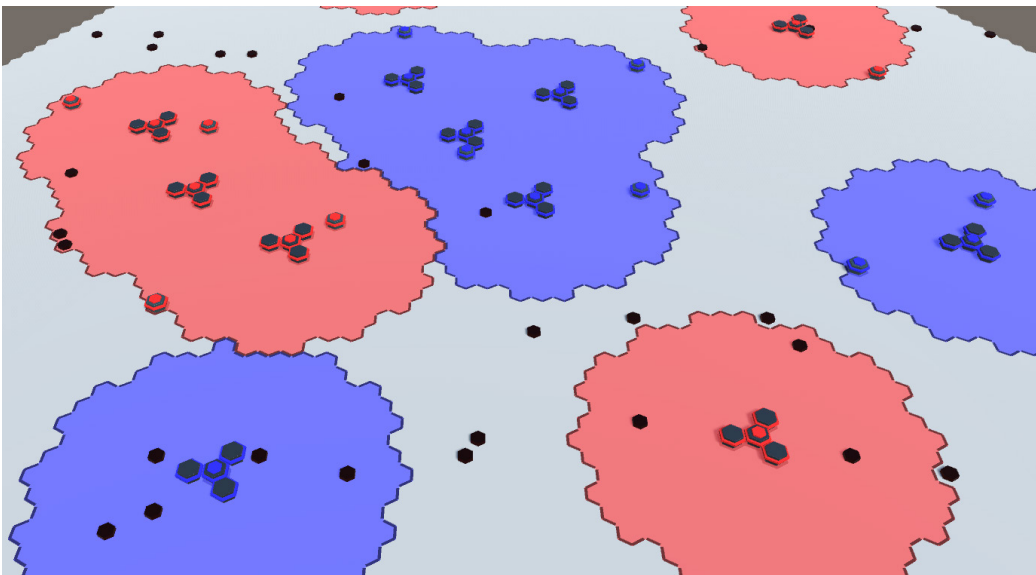


FIGURE 24
Digital Prototype

2.4.4 Version 1 – Evaluation

The most important finding of this first version of the game was that it already worked in such a simple state. The core loop of expansion and resource gathering functioned very well and gave great rhythm to the gameplay. Both core mechanics also seemed to trigger the players' innate desire to control, collect and possess. In general, the gameplay experience was slow and cerebral, like a game of chess. Players took a lot of time to think and strategize about their moves and there were already some interesting decisions to make. I wasn't sure whether I liked the slow pace of the game, but, of course, it was great to see that players already had to put a lot of thought into the gameplay. Naturally, the game still had a lot of issues, but overall I found myself in the optimal situation for the start of a game project – at the core, both the game system and the player engagement were already at a good level.

Area Control

Area control was the most dominant mechanism of the game, as it was the first action players took and stayed the center of strategic thinking throughout gameplay. Players thought about where to expand to and how to cut off or encircle their opponent and the evolving territory structures were also visually intriguing.

The most interesting spacial conflicts would happen where opponents' territories came close or even collided and resource deposits were contested. These frontlines, however, didn't form too often. In general, it was more rewarding to expand into open areas rather than engaging contested spaces. The main reason was a significant first mover advantage, as whoever got into an area first had the better chance to gather the free resource deposits. Additionally, the extra range of the gathering buildings acted as a spacial buffer that prevented actual territories from colliding and, most importantly, once resources in contested spaces were either controlled or gathered the tension was instantly over. All in all, there weren't many interesting actions to take on a frontline.

The expansion into open areas worked great though. There was a constant pressure to compete for unconquered spaces and players grew their territory to cut off their opponent. Opponents could respond by paying the extra price to jump over the encircling area and cut it off from the other side. This competitive expansion process created a great back and forth dynamic between the players and lasted throughout most of the game.

Multi-Purpose Resource

One apparent issue was the multi-purpose nature of the basic resource. On the one hand it served as money to purchase buildings, but, on the other hand, it was also the highscore. This unclear role confused some players and they found it irritating to spend their score points to buy buildings. Furthermore, it was often hard to tell which player was ahead because the amount of resources – which the players thought of as the scoreboard – fluctuated so much. As a game designer, multi-purpose mechanics are generally something to strive for, as nothing is more elegant than solving multiple problems with one solution. But when the different purposes of a gameplay mechanic don't align, it can be detrimental to the player's experience and understanding of the game and should therefore be avoided.

Uniformity

Another problem arising from the simplicity of the game system was a general feeling of uniformity, as the gameplay experience was very constant and never really changed. The game didn't have any specific beats or milestones, no altering short term goals to create some variance and the map and distributed resource deposits didn't have any special hotspots. The suspense curve of the game lacked any significant turbulence. The game worked well, but it felt a little bland.

Late Game

The only gameplay phase that deviated from the uniform pacing was the late game. Sadly, it did so in a negative fashion. Once most spaces had been controlled, conflict completely died down and players just took turns to gather the leftover resource deposits scattered over the map. This resulted in a boring, predictable and anticlimactic finish to the game. At this stage of development, I wasn't sure if this was a fundamental problem of the game system or just a result of the specific end condition I had defined. In any case, it was clear that I had to work this issue out as soon as possible.

2.4.5 Workflow Evaluation and Outlook

Overall I was very satisfied with my workflow to this point. The initial ideation provided me with great points to start from and I had a good vision of what game I wanted to create. The analysis of other strategy games, however, gave me a rather distorted notion of how complex my game would have to be. While this analytical approach didn't turn out to be a good way to conceptualize the minimum viable prototype, it at least gave me a lot of knowledge and understanding of my design space. I'm very glad that I radically changed my design method and started working on a paper prototype, as the simplicity of the technique allowed me to iterate quickly and focus on the core design elements. It took only a few hours to create a playable game, which proved convincingly that it wouldn't need multiple resources and production chains to generate a compelling gameplay experience.

I was very happy to achieve such a quick and successful first iteration of my game and was convinced that the core system had a lot of potential for improvement. Because the game was still so simple and some of its issues were very obvious, I decided to directly start the next design iteration without working on an AI opponent. I knew the game would still need to change and thought that AI development, at this point, would be a waste of time.

2.5 Version 2 – Core Prototype

After finishing the first full cycle of the design process, it was time to start the second iteration. My main approach was to come up with solutions for the most glaring issues of the previous version. I wanted to focus on a clearer gameplay goal and end condition, the reduction of uniformity and an improved performance feedback.

2.5.1 Game Design

Core Resource

Many issues of Version 1 pointed to the same solution – the game needed another resource. I wanted to have a distinct highscore resource that wasn't drained by construction costs, which gave me the opportunity to insert one of my key gameplay ideas – the production pyramid – into the game. The special resource would be produced from the standard resource by constructing a unique production building. This alone should make the game system more complex, interesting and less uniform, but I also saw another potential purpose for the special resource – it would be used in the construction of area control buildings, which would mean that the number of special resources limits a player's ability to expand into new areas. Hence, players must produce special resources not only for the final score, but also in order to enable further expansion early in the game.

The addition of the special resource resulted in an interesting evolution of the core gameplay loop, as the two step cycle of Version 1 was now extended by a third step in the form of production. Players would expand their territory to gather standard resources, which were consumed to produce special resources in order to further expand territory. I named the special resources cores to emphasize their importance.

Dismantling

Interestingly, the fact that cores would control the expansion of territory had the potential to synergize greatly with another initial gameplay idea of mine – the dismantling of buildings. When a building was dismantled, players got back some of the construction resources, so dismantling an area control building would give back the core that was used in its construction. This means players now had multiple options to gain a core, which made the game loop even more interesting (**Figure 25**). A crucial part of this system was that it fixed the main problem of the multi-purpose resource in Version 1. As spent Cores can always be regained by dismantling, they can never be truly lost, which makes them easier to understand and track as a highscore representation.

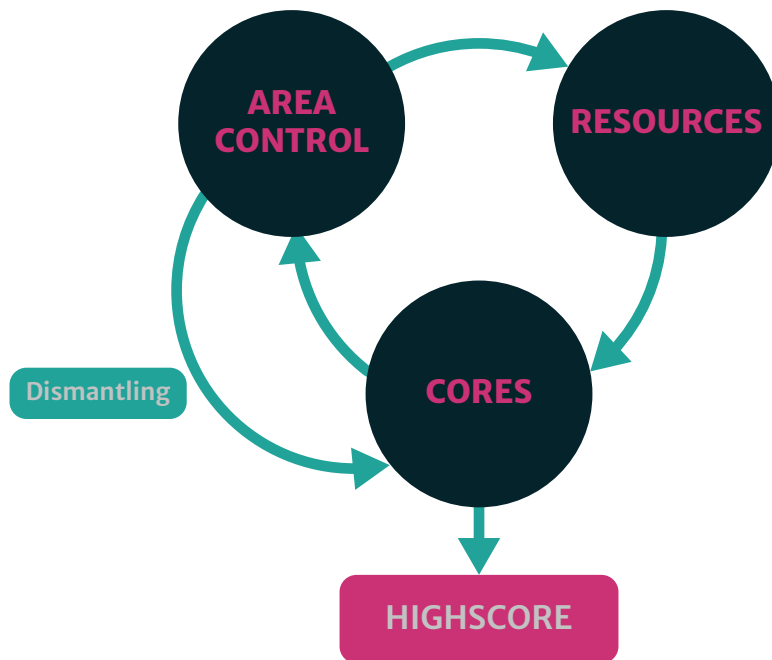


FIGURE 25
Enhanced
Game Loop

Gameplay Goal

I changed the end condition of the game to a fixed number of turns. Where the game tended to fizzle out before, it was now concluding at a specific determined point. This worked well with the dismantling mechanism, as players now had to think about when to start dismantling their last area control buildings to get back the built-in cores. In order to ensure absolute clarity about the gameplay goal, both important rules for the end game – the fixed number of turns and the win condition of having the most cores – were displayed on a tutorial screen at the start of the game (Figure 26).

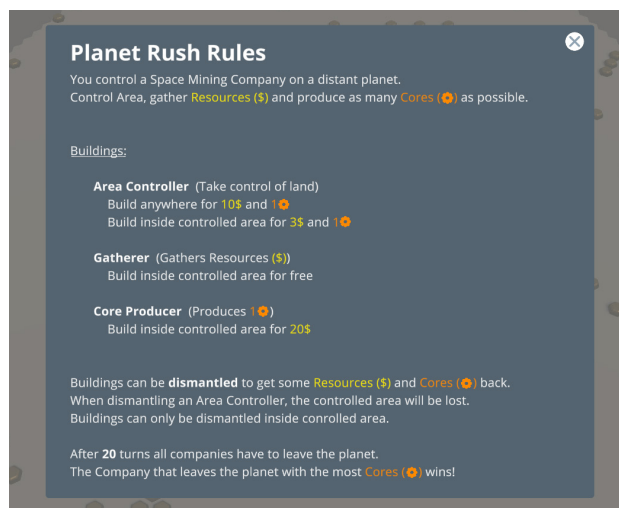
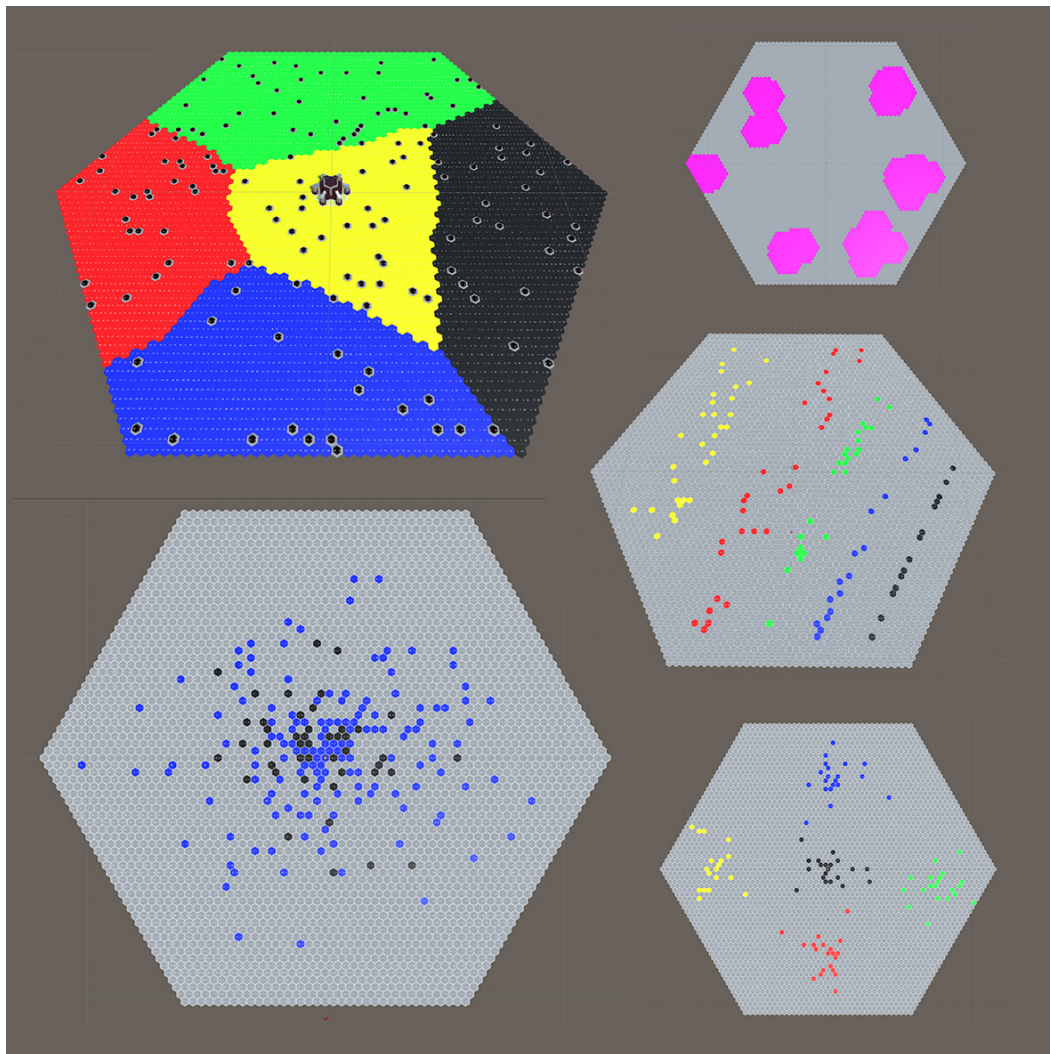


FIGURE 26
Tutorial Screen

Map Generation

I knew that map design would be critical for my game because a simple change of resource distribution could result in very different gameplay. At this phase of the development, I thought it would make the most sense to have a procedurally generated map to provide variety to separate playthroughs. I put a lot of work into developing a modular set of spacial design methods. In the end, I was able to create multiple types of zones and distributions like fuzzy lines and gaussian clusters (Figure 27). After some experimentation I ended up with a quite elaborate map generation that was based on a branching tree structure that grew from the center of the map (Figure 28). I distributed the resource deposits over the branches of the tree and generated some compact clusters on some of the nodes, which resulted in a nice balance of crowded and empty spaces. For this version of the game, the procedural map generation was sufficient, but it already proved difficult to achieve the required consistency to guarantee consistent results.

FIGURE 27
Visualizations
of procedural map
generation
techniques



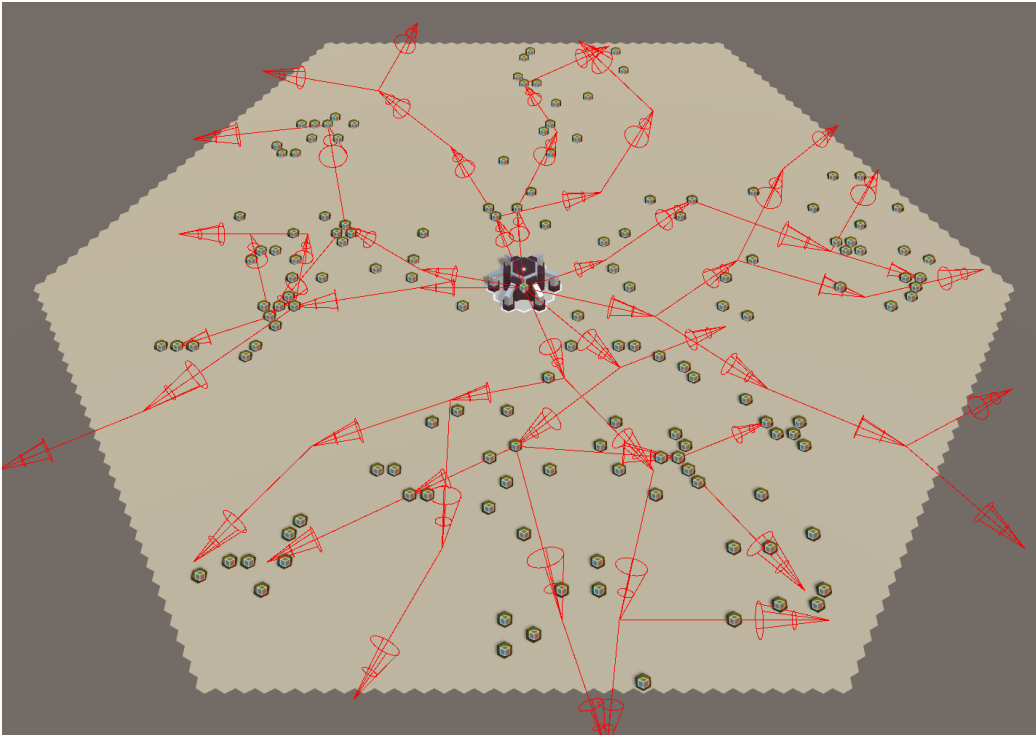


FIGURE 28
Branching tree
structure of
the final map
generation

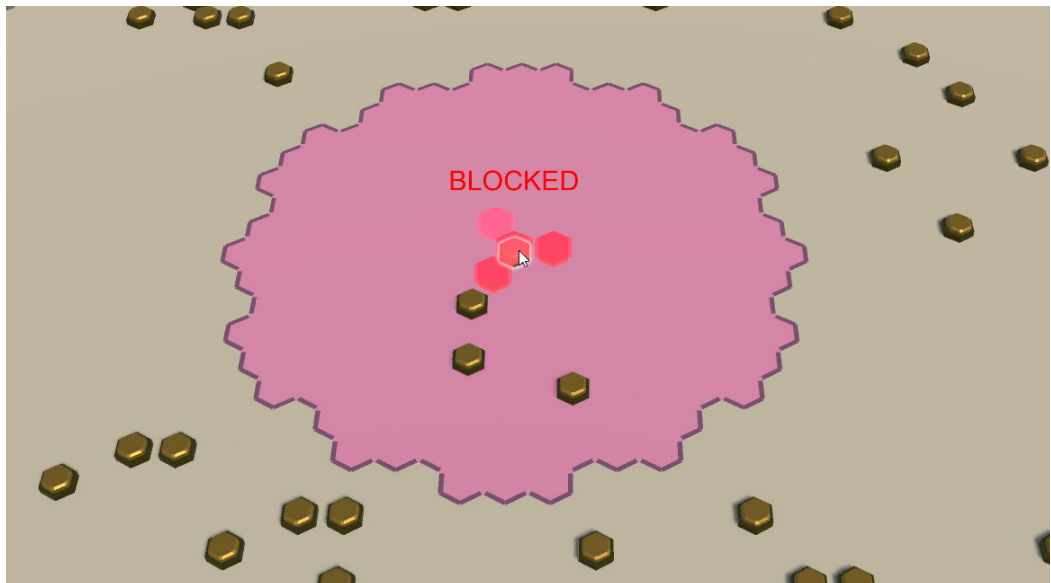
2.5.2 Visual Communication

As stated before, I want the focus of this paper to be on game system design. Therefore, I don't want to go into too much detail about the visual design I did throughout this project. However, some visual elements are absolutely essential in communicating the actual gameplay and it became apparent to me, at this stage of development, that a game with discrete mechanics actually heavily relies on visual communication to be comprehensible. A game that implements physical movement can be easily understood by watching objects move through space, but a game with more abstract mechanisms has to make use of more abstract visual techniques such as graphical user interfaces. I definitely underestimated the amount of work required to effectively communicate the activity of the player, mouse interactions and simple gameplay logic. In the three step workflow of designing a mechanism, programming its logic and creating its visual communication, the visual step would often take the most time. Having to spend so much time on the development aspect I was least interested in definitely caused me some frustration and certainly made me miss the times when I had artists in my development team.

In essence, visual methods have to be utilized to communicate what is happening or what will happen in the game. For example, when searching for a position to place a building, the game showed a simulated ghost building on the map, which always tells the player where the building would be constructed if the mouse button was clicked.

When the placement was not possible at the current position, this had to be communicated as well. I orientated myself on other building games and implemented a building model with a transparent red material for this situation (**Figure 29**). Of course, the ranges of buildings also had to be displayed when in placement mode, which I did by procedurally creating area and border meshes. For the gathering building, however, this didn't suffice. Without additional visual help, it wasn't clear to players that placing a gathering building would interact with the resource deposits in range, but when I implemented a shiny golden outline around the resource deposits it became understandable immediately. It not only clearly communicated what would happen when placing the gathering building at this position, it also implicitly taught players that resource deposits controlled by the opponent can't be gathered from, as those wouldn't show the golden outline (**Figure 30**). Furthermore, I added a visual indicator of the number of resources players would gather, which improved the understanding and flow of the game even more. This number would also rise up for a few seconds after the gather building was placed to assure players that everything worked as expected. Combined with the disappearance of the gathered resource deposits and the updated number of resources in the inventory panel this successfully communicated the entire logic of the mechanic.

FIGURE 29
Red ghost building



In contrast, a mechanic that was very difficult to communicate was the difference of construction costs for building the area control building either inside of controlled territory or anywhere else on the map. I displayed the construction cost right next to the ghost building, but players wouldn't register the difference. Often players would place an area control building outside of their territory and not realize the high cost they had just paid. Additionally, the displayed construction cost would also turn red

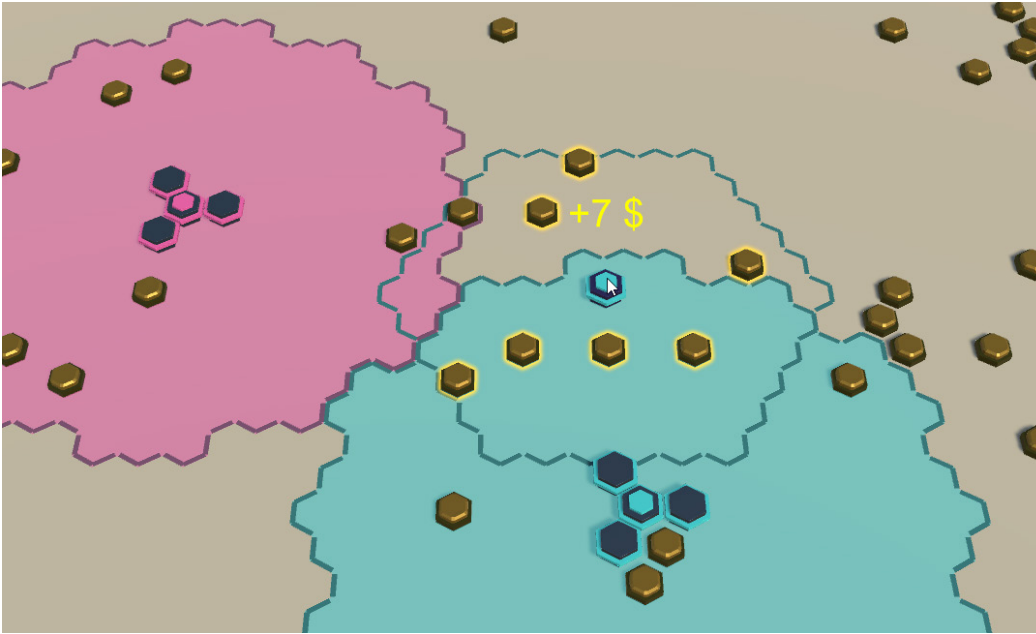


FIGURE 30
Resource gathering
visualization

when the required resources weren't available and I was sure that all the red visuals would effectively communicate that the placement was impossible and why (Figure 31), but most players would spend a lot of time searching for a great position for the building and only realize that construction wasn't possible once they actually tried to place it. It seemed that players were so occupied with finding a good placement position that they ignored all other visual cues at the time. Therefore, the information that construction isn't possible had to be conveyed before players engaged in their placement search, which I achieved by changing the appearance of the building button, which had to be clicked before entering placement mode.



FIGURE 31
Red world text
indicating missing
resources

2.5.3 Playtesting

Once the main functionalities were implemented I entered another phase of rapid iteration. This time, I incorporated a lot more playtests with other players because I really wanted to find out if players intuitively understand the new game loop. I wanted the game to effectively explain itself. In the span of roughly two days, I would playtest and evaluate the current state of the game and then do adjustments for the next playtest. I repeated this cycle about five times. In the first playtest, I found out about very obvious issues, such as logic bugs, missing explanations and confusing controls. Once these were fixed, I could focus on more subtle comprehension issues and balancing gameplay parameters. After multiple consecutive improvement passes, the game reached a good level of polish and players didn't encounter any more issues with user experience (UX) or general comprehension. Now they could focus entirely on the strategic gameplay. Overall, player feedback was of great help to fine-tune both the game design and visual communication and I was able to produce a great final prototype for Version 2 of the game.

2.5.4 Version 2 – Evaluation

Version 2 of the game was definitely a great improvement. I managed to extend the system with a few precise mechanics, which effectively alleviated some of the earlier problems and the added complexity provided more depth to the gameplay without creating any new issues. While the game remained relatively simple, it now afforded more actions to take and more interesting decisions to make. The gameplay goal and end condition were clearer and the added UX improvements made the gameplay experience smoother and more enjoyable.

Core Resource

I was especially surprised by how well the core resource fit into the game system. It not only solved multiple issues, but also created new opportunities. In contrast to the standard resource of Version 1, the core resource really was a successful multi-purpose mechanic. To begin with, it was a far better highscore indicator, as players could track their exact score by adding the number of cores inside their area control buildings and the number of cores in their inventory. It was now also a lot easier to estimate which player was generally ahead. Furthermore, the core resource seamlessly added another step to the core gameplay loop and also allowed for the dismantling mechanic to be introduced. Attaining the core resource was now one more goal for players to strive for, which resulted in a more compelling gameplay progression and the two different ways to gain a core provided a unique interesting choice to make.

The fact that territory expansion was now only possible by using the core resource created an interesting design situation. The core resource now effectively controlled the gameplay system of area control, which meant I could use it as a precise design

tool to affect this single subsystem of the game. If I wanted to make expansion more challenging or incentivize dismantling area control buildings, I could do so by making cores rarer or more expensive. Because the core resource only affected the area control system, such balance changes would have little to no effect on other systems. This arrangement, where different resources manage separate gameplay subsystems, is definitely a great way to increase design control and, in addition, it often makes these systems more accessible and easier to comprehend for the player as well.

Dismantling

The dismantling mechanic generally integrated nicely into the gameplay. Players were now forced to dismantle area controllers and give up parts of their territories to progress into new areas. This created some nice strategic dynamics. Instead of waiting to gather from save resource deposits until the end of the game, players would now quickly deplete all resource deposits in the vicinity of one area control building in order to dismantle it. Additionally, the core production buildings could be dismantled for a significant amount of standard resources, but only if they remained inside a player's territory, which meant that players had to think very carefully when looking for suitable placement positions. Overall, the dismantling mechanic asked the players to do a lot more strategic forward thinking.

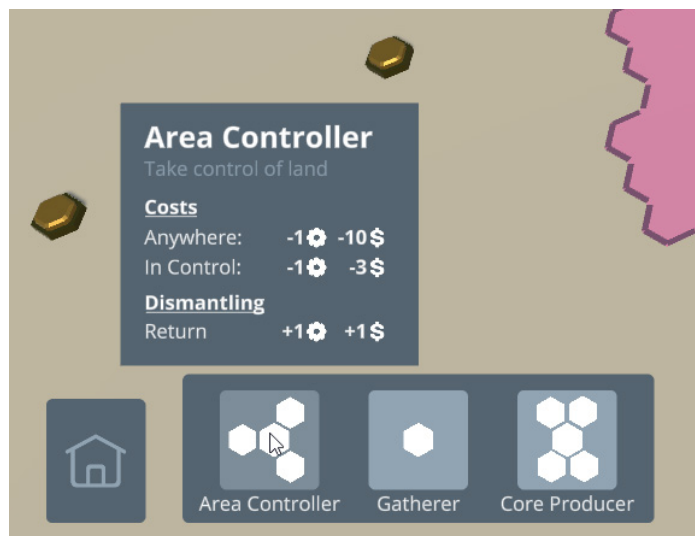
While the dismantling mechanic worked very well systemically, it also became clear that players had some trouble with the general idea. Even though the tutorial screen mentioned it, multiple playtesters just forgot about it after the early game and when they had no more cores to expand, they didn't think about dismantling as an option at all. This, of course, was mainly a communication issue. The dismantling button itself was displayed in the building window, which was shown when a building was selected (Figure 32), but because players never had to select buildings for any other reason, they



FIGURE 32
Dismantling button
inside the building
window

never did. I had to find a better way to constantly remind players of the dismantling mechanic and adding the dismantling return values in the small context windows of the building buttons helped a little bit (Figure 33). These reminders could have been improved further, but the main issue, I think, was that players just aren't accustomed to such a strong focus on dismantling in a game about placing buildings. In most building games the goal is to construct a big, complex structure – the larger, the better. Dismantling is only utilized to fix some minor mistakes here and there, so the idea that dismantling is essential to the game's progression was unintuitive for most players. However, once they understood the dynamic and got used to it, they accepted it and even found it very interesting and original.

FIGURE 33
information about
the dismantling
return inside the
building type
context window



Predictability

While Version 2 of the game was more complex than before and provided more options for players, most of the time, there was still a quite obvious best action to take, which resulted in a relatively high level of predictability. In the early game, the first dozen moves or so often followed a similar pattern and the most standard strategy (Jump, Gather, Jump, Gather, Gather, Produce, Gather, Jump) was used by players more than half of the time (Figure 34). This was only rarely modified with different moves when the map generation allowed for it.

The end game was even more predictable. Of course, players would want to dismantle all their remaining area control buildings to get back their valuable cores, which meant that from a certain point, the sequence of remaining moves was basically determined. This point usually happened with around 5 to ten turns left, depending on the players. While the end game was more interesting than before, its predictability still made it quite anticlimactic and once players knew how the game would end, all tension was lost and it felt meaningless to continue to play.

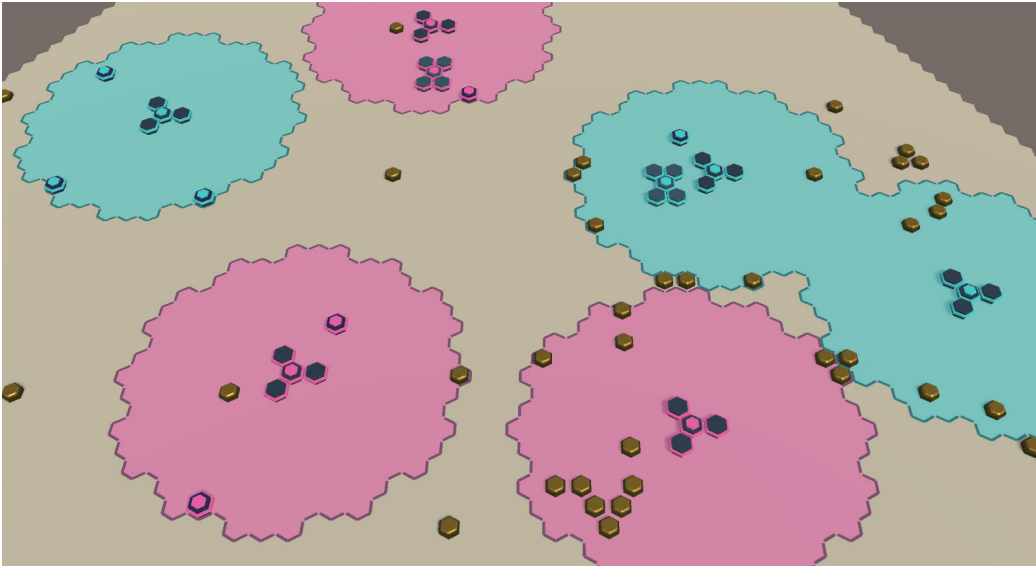


FIGURE 34
Game state after
two players played
the most common
sequence of actions

Furthermore, because the gameplay lacked any potent comeback mechanics, players that were clearly behind didn't feel like they still had a realistic chance of winning the game, which, of course, killed their motivation to play on completely.

Sadly, the general predictability of the game resulted in a lot of players not wanting to play too many sessions. After two or three playthroughs, players felt like they had experienced everything there is to see and couldn't think of any new strategic approaches to play the game.

Spacial Conflict

As discussed before, Version 1 of the game didn't incentivize spacial conflict. As I didn't really do anything to specifically address this issue in the second iteration, it basically remained unchanged. The absolute nature of area control and resource deposit gathering still resulted in a strong first mover advantage that disincentivized engaging in territorial disputes.

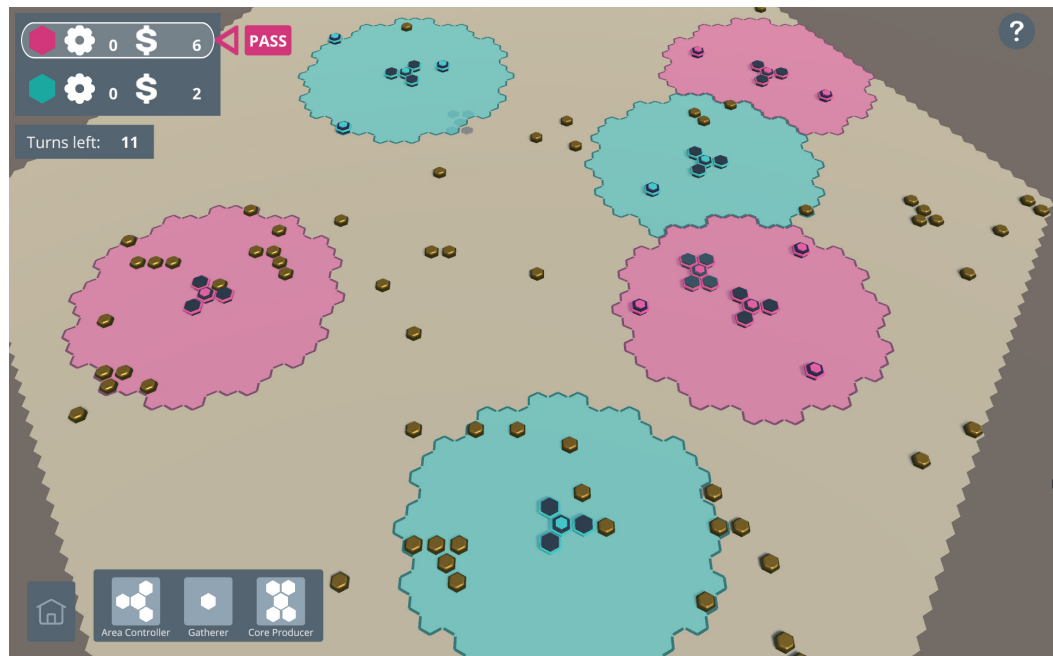
However, I realized that the map design also had a great impact on how players would approach expanding into free or contested areas. Generally, if there was an abundance of resource deposits on the map it was strategically superior to expand to open spaces. Players just wouldn't fight over a few resources if there were plenty of opportunities elsewhere. On the flipside, compact patches of resource deposits with a relatively large size would actually encourage conflict. When one player covered roughly half of a valuable patch, the opponent would jump in and cover the other half. This move was viable because the second player gained a lot and set back their opponent at the same time.

The multiple playtests showed that players still had a strong urge to engage in aggressive territorial conflict and it was clear to me that the next design iteration would have to focus heavily on improving this gameplay dynamic.

2.5.5 Workflow Evaluation and Outlook

The second design iteration of this project was a great success. I had an excellent design phase focused on the evaluation findings and issues of the prior version and then successfully prototyped and playtested the game in multiple smaller iteration cycles. I managed to keep the scope of new design elements relatively small, which made the whole process fast and controllable. I always had the feeling of fully understanding the system as a whole and the impact that specific alterations had on it, which made the design process very precise and ultimately successful. Overall, I made a lot of good progress and Version 2 was already a good functioning game. Now, I felt, was the right time to turn my attention to the second part of this project – it was time to begin the development of the Game AI.

FIGURE 35
Final prototype of
Version 2



2.6 Version 2 – AI Development

I wanted the AI player to function exactly like a human player. It would wait for its turn, make a decision and make a single move. Ideally the moves would be clever and never noticeably stupid. Because I was generally planning on developing more iterations of the game design and game AI, I wanted to implement an AI system that would be able to scale and adapt with a more complex game and decision making process.

2.6.1 AI Architecture

One of the first steps in developing game AI is to decide on the right AI architecture. In general, most AI systems do the same thing – they make decisions for behavior execution according to the game’s state – but their fundamental structure can differ significantly. The most common types of AI architecture are Scripted AI, State Machines, Behavior Trees, Planners, Utility Systems and Neural Networks (Mark, 2012). As I wanted to work with the right system, I spent a lot of time doing research on the different approaches, but sadly, it’s very difficult to find good information about the game AI of commercial strategy games and most material online is either about other genres or very superficial. Even books on game AI only cover mostly abstract examples. Apparently, AI for strategy games is just too complex to communicate concisely in a reasonable scope. It is clear, however, that there isn’t a one-fits-all solution, as each game plays differently and requires a custom-built AI system.

Prior Experiences

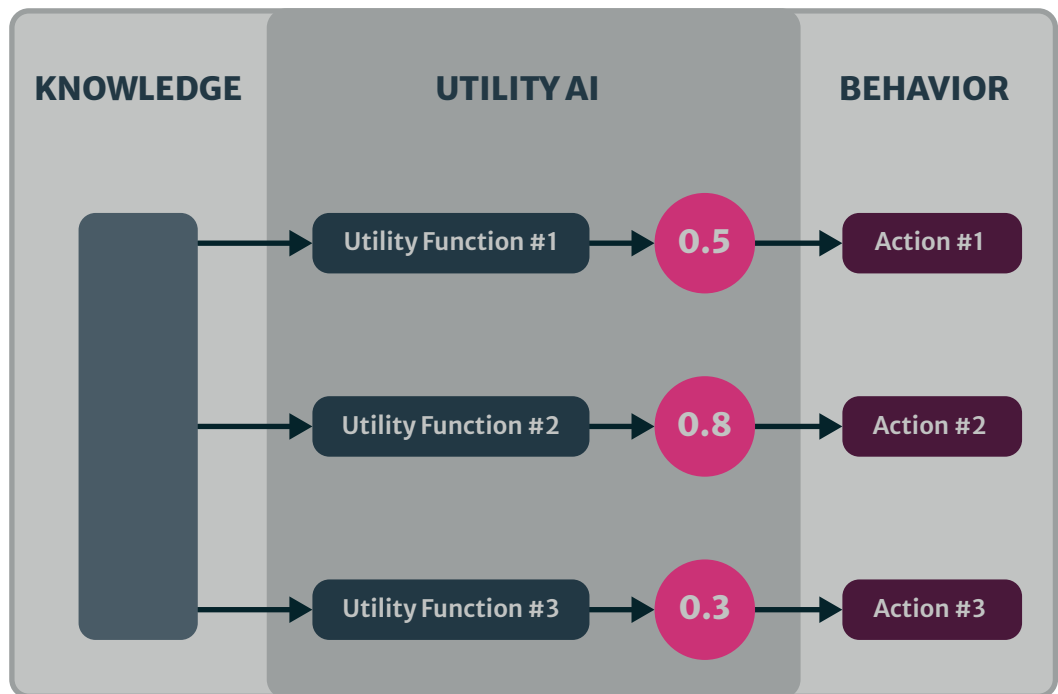
I had some prior practical experience with most of the AI architectures, except for the Utility and Neural Network types. Scripted AI works well for simple use cases but doesn’t scale very well. State Machines and Behavior Trees are relatively easy to understand and implement and afford a lot of design control, but are also somewhat rigid and might struggle with the complexity of a dynamic strategy game system. Planners, on the other hand, are supposed to work well within dynamic systems. In the 6th semester, I did a small project where I implemented a Goal Oriented Action Planning (GOAP) AI, however, in my case, GOAP actually didn’t scale well when tasked to make decisions in a complex economy system. I don’t want to discuss this in detail again, but inside a complex, dynamic system, the Planner needs to factor in too many possibilities and the game state, the actions and the goals are difficult to abstract. Calculated action plans are often not agreeable with all AI goals and when the game state changes, plans can become invalid and behavior execution has to be stopped and replanned. In the end, the planning mechanism can result in a game AI that appears undecisive and incompetent (Thompson, 2018).

Utility AI

One AI system I never worked with was Utility AI. In this architecture the Game AI lists all possible actions and scores them with individual utility functions (**Figure 36**). The highest scored action is then executed. Input for the utility functions can be the game state or specifically generated data, generally referred to as knowledge representation (Brewer & Graham, 2018). The strength of Utility AI is its flexibility, as it doesn't have pre-defined behavior routines and transitions like state machines or behavior trees. Therefore, it can adapt to very complex, dynamic environments. Whatever the state of the game, Utility AI will score the available actions and execute the most viable. On the flipside, the architecture concedes some design control, as there are no definitive trigger and effect relationships. Instead, utility functions must be carefully configured to execute reasonable behavior in all situations and when the behavior becomes more complex, it might be increasingly difficult to debug and comprehend why the AI makes certain decisions.

I really liked the concept of Utility AI, as it promised flexibility, the ability to handle complex situations, life-like emergent behavior and the ability to scale well. I also thought the mathematical nature of the utility functions and the design approach of configuration really suited my personal preferences and skills. Most importantly, I was hopeful that Utility AI, in contrast to Planning AI, would be able to handle complex economic values without any problems. Of course, I didn't know for sure whether Utility AI would work well for my game, but I was optimistic and wanted to give it a try. In the worst case, I would still gather valuable insights into an interesting AI architecture.

FIGURE 36
Basic Utility AI
architecture



2.6.2 Decision Making Analysis

As discussed before, the Game AI was meant to mirror human players. Therefore, it made sense to start the AI design process with a rigorous analysis of how humans actually play the game. I wanted to find out how exactly decisions were made and which information was processed in order to translate this into the technical implementation of the AI player.

Utility

The most important question in the decision making was how valuable an option was in the context of the gameplay goal. As a player considered where to place a resource gatherer, a placement which resulted in more gathered resources was, of course, more valuable and was therefore prioritized. However, often more than one information was factored into the viability of an action. Maybe some resource deposits were contested, as the opponent could gather them in their next turn. Therefore, gathering from the contested resources – and snatching them away from the opponent – had more value and priority than gathering from safe resources. According to utility theory this usefulness of a particular action can be given a singular numeric value, which is referred to as utility (Graham, 2014).

Time Horizons

Another crucial part of understanding decision making is to look at time horizons. The simplest decisions – like where to place a gatherer to maximize resources – are made with only the next action in mind. Once the gatherer is placed, the resources are added to the player's inventory and the extent of the decisions ends.

A similar but slightly more forward-looking decision is made when a player thinks about whether they can do something else after placing the gatherer, like whether the resources added to the inventory would allow the player to afford placing an area controller in the next turn. In this case, the time horizon of the decision spans over two turns. In this way, players can formulate multi-step action plans to reach certain goals. A player might grow their territory by placing an area controller, in order to place two resource gatherers, in order to afford their ultimate goal – placing a core producer.

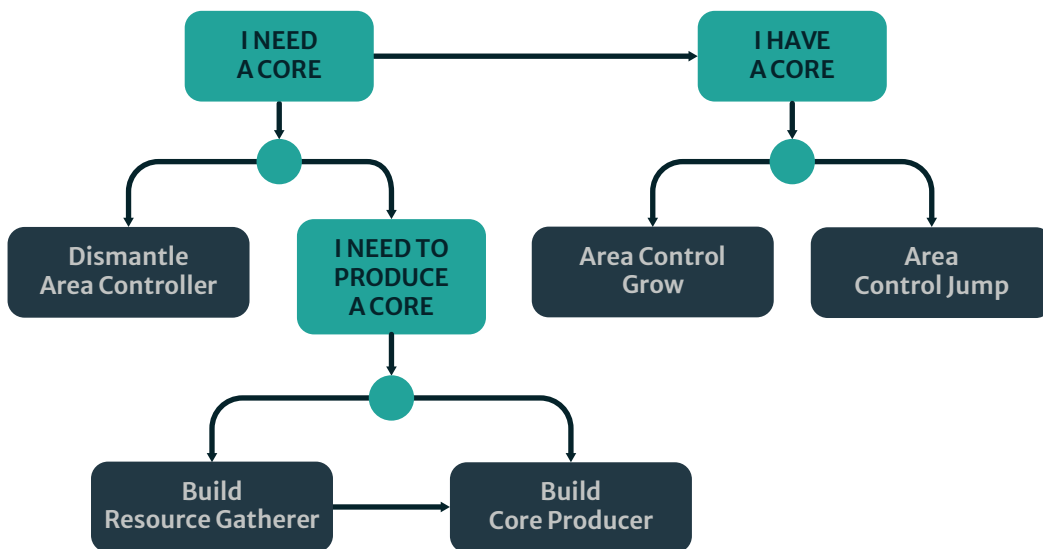
However, planning doesn't always involve such linear paths towards a distinct goal. In complex strategy games players often think about what possibility spaces can be opened up by specific actions. When a player decides on placing an area controller into an open space, they might think about a number of different actions that can be taken afterwards – gathering, producing, growing into nearby areas –, which can be executed in a variety of different sequences. In this case, the quality of the prospective possibility space determines the utility of the immediate action.

Modeling Decision Making

While analyzing the decision making process of the current game version, I realized that it can be modeled in a variety of different ways. Not surprisingly these models corresponded with the common AI architectures, which all have slightly different approaches to handling utility and time horizons.

One of my models heavily resembled behavior trees (Figure 37), as the decision was made by going through a tree structure consisting of junctions and sequences. For example, players would ask themselves whether they have a core and if they didn't, they had to choose between two options – dismantling or production. While the dismantling option was a single action, the production option would go through a two step sequence of gathering enough resources and then placing a core producer. This tree structure approach, however, didn't consider some of the crucial details of the decision making process, like how viable the dismantling of a particular area controller was or how many resources were still needed to afford a core producer. The behavior tree architecture just prioritizes one option over the other instead of considering subtle nuances.

FIGURE 37
Behavior Tree
decision model



Often human players would aim at a goal and come up with an appropriate action sequence to satisfy it (Figure 38). A player would want to expand into a lucrative area, which would require a core. This would require building a core producer, which in turn required resources. Therefore the player would decide to place a resource gatherer. These dependency chains are, of course, the logical basis for Planner AI.

Often, however, players didn't have a distinct goal in mind and just decided on actions that moved them in the right direction. Some look-ahead AI methods like the minimax

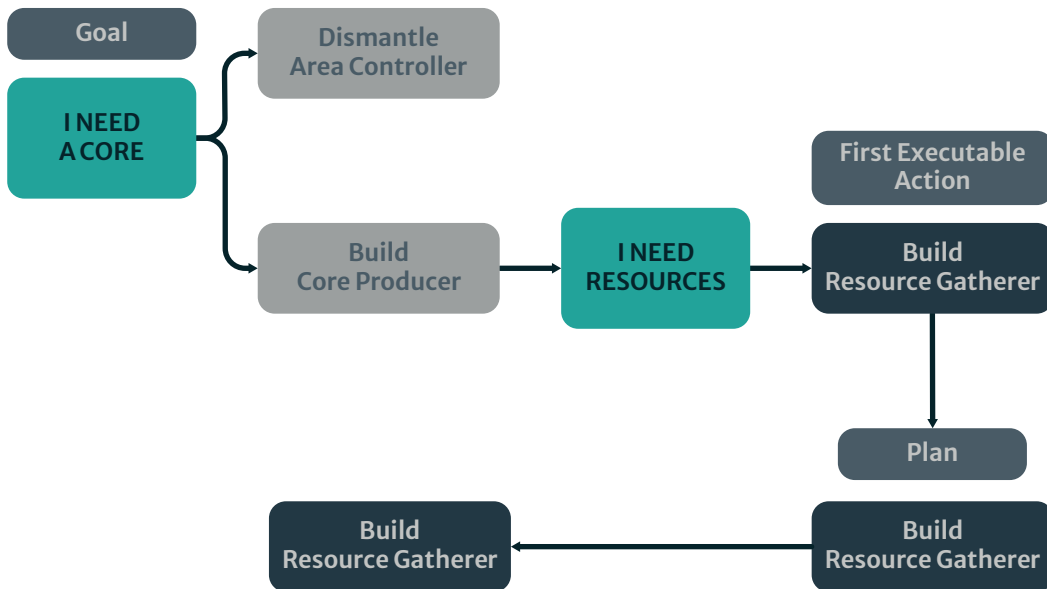


FIGURE 38
Planning decision
model

algorithm come close to such an opportunistic playstyle. The algorithm gives a utility score to all possible consecutive game states and executes the most valuable branch of the tree. While this decision making method can be very powerful, the tree structure of the possibility space is generally very expensive to calculate and quickly reaches its limit with increasing complexity of the game system.

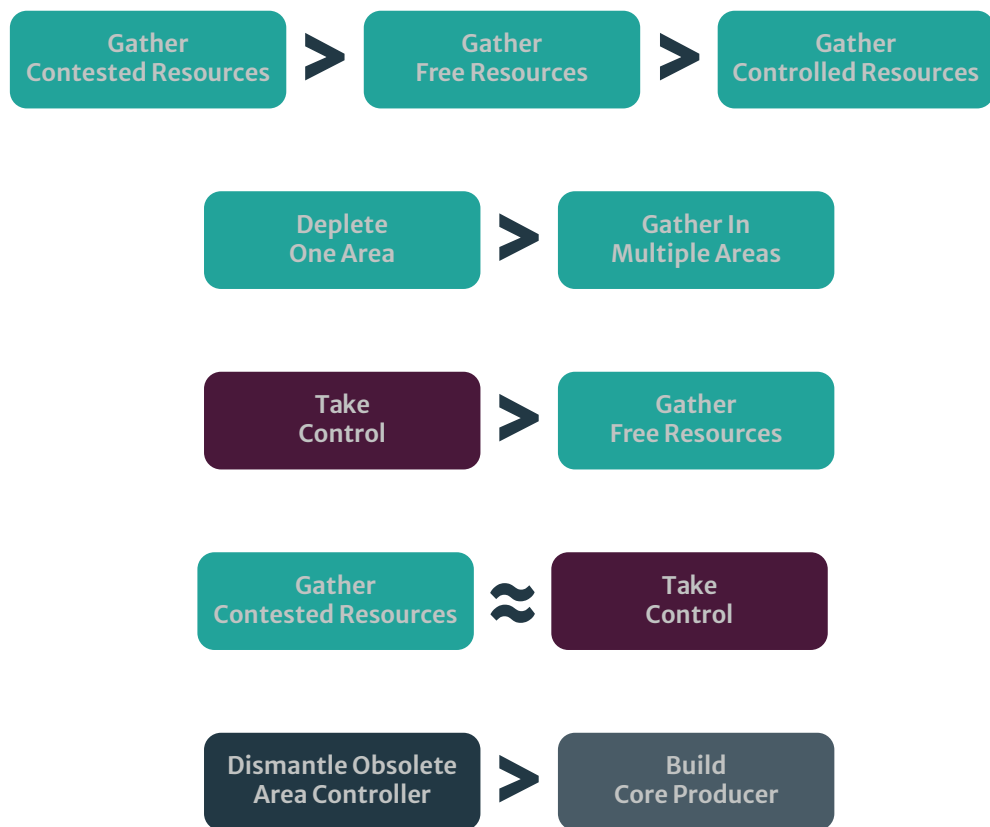
Utility AI Modeling

Interestingly, the decision making process of Utility AI isn't particularly concerned with action sequences or future possibility spaces. Fundamentally, it just scores each possible action in the context of the current state of the game. But human players did think ahead when playing the game, so the big question for me was how Utility AI could model this forward thinking. The answer was actually pretty straight-forward in practice – the utility functions needed to factor in the future potential of their action in some way. In many cases this potential could be easily quantified. For example, when calculating the utility of placing an area controller at a certain position, the utility function would factor in how many resource deposits were in the vicinity. The more resource deposits were around, the better the score was for a specific position. With this model, the decision of placing an area controller the current turn already factored in possible next turn actions of placing resource gatherers.

Even more exact dependencies between actions could be considered by utility functions. When scoring a position for placing a resource gatherer, a utility function could factor in the amount of resources in the inventory afterwards and increase the score if that amount is enough to pay for a core producer. With such a consideration the core producer is basically an implicit goal and an action that enables its construction is scored higher accordingly.

When looking at the decision making process of the game, I could see an interesting pattern – there were a number of priority principles (Figure 39). For example, acquiring area control was generally prioritized over gathering resources because players would rather claim new territory first if they could afford it and gather more resources later. Furthermore, gathering from contested resource deposits was always preferred to gathering from uncontested or even controlled resource deposits and when in need of a core, dismantling an obsolete area controller had priority over constructing a core producer because it was more cost effective.

FIGURE 39
Decision making
priorities



I realized that the actions could be arranged by priority to create a staircase structure (Figure 40). An action would be positioned within the hierarchy by an appropriate base score and further value could be added for specific properties to compare two actions of the same type, such as comparing two positions to place a resource gatherer. In practice, this should lead to a system in which executing one action would enable another higher step action in the staircase. For example, gathering resources would enable building a core producer, which would enable building an area controller. With this staircase design pattern, I would hopefully be able to effectively imply the observed action sequences of the current gameplay.

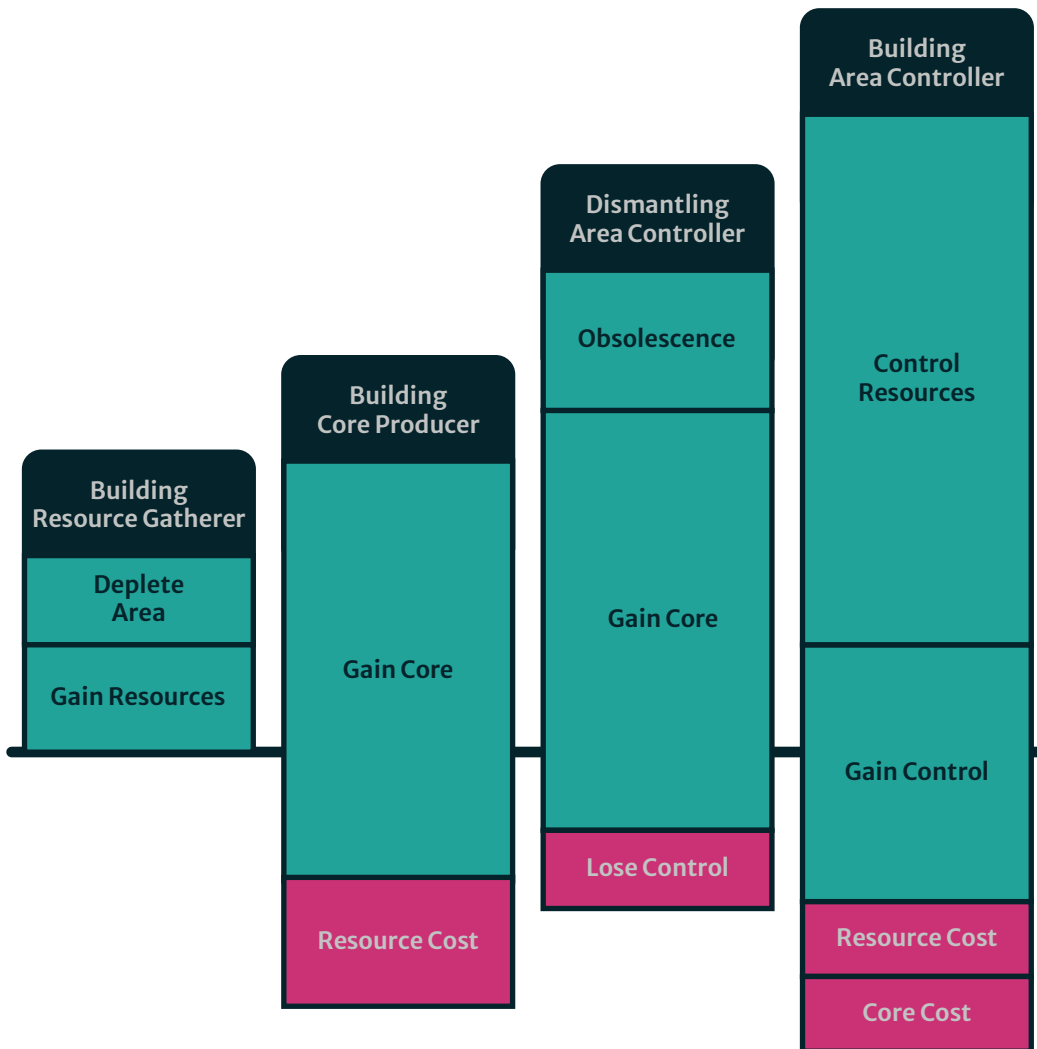


FIGURE 40
Decision making
staircase pattern

2.6.3 Utility AI Design and Implementation

After the extensive decision making analysis, I was sure I could model the decision making process with utility functions, but first I had to technically implement a Utility AI system in Unity. Game AI generally consists of three layers – knowledge representation, decision making and behavior. As discussed before, knowledge representation is the data that feeds into the decision making process and I was confident that the pure game state and a good influence map system would cover this aspect sufficiently at this point. Furthermore, the actual behavior was already precisely defined in the game system and the company class contained executive functions for all possible actions that just needed to be called by the AI player. This left me with only one real implementation challenge – the decision making architecture.

I looked at a few open source Utility AIs like Apex Utility AI and ReflexityAI. I was intrigued by how different they were, both on the side of fundamental decision making logic and on the side of technical implementation in Unity. On one hand, most of what I saw was quite sophisticated at the level of technical implementation, but relatively simplistic in actual decision making logic. I was really missing some of the features described in Behavioral Mathematics for Game AI (Mark, 2009), like modular considerations and response curves. I definitely learned a lot by analysing these Utility AI systems, but in the end I decided to develop my own custom implementation.

Decision Making Architecture

A general goal of any system should be modularity. A modular game AI system provides the designer a flexible toolset that allows for quick and effective development of AI behavior. One system I was really impressed with was the Infinite Axis Utility AI, in which every decision factor is processed in modular considerations, which have inputs that map to normalized outputs via response curves (Lewis & Mark, 2015). Each individual action is scored by a group of parallel considerations, which makes the system modular, elegant and easy to work with. However, when trying to apply it to my game, I realized that it didn't really fit my specific requirements. The biggest issue was that the Infinite Axis method creates action variants for each possible action configuration. For example, in a RPG style battle, a character might be able to use their standard melee attack on four different enemies. The AI system would then create four unique action variants and score them individually. In my game, however, a building could be placed on a huge number of different tiles, which meant that an approach that considers every possible action would result in thousands of actions that would need to be scored. It became clear to me that there was no universal Utility AI system and realized that I had to create a custom version for the specific requirements of my game.

After looking at so many different types of Utility AI implementations, I decided to just develop a simple architecture that fits my game well, even if it meant conceding some modularity. At the current complexity of the game, this should be adequate and I could always adapt later on. The highest entity of the decision making architecture was the Utility AI Brain (**Figure 4.1**). An AI player would tell it's brain to execute at the start of its turn and the brain would then tell each action to score itself and execute the highest scored action. Instead of using modular considerations, I created a custom class for each action with hard-coded utility functions, which were added to gameobjects as components and the decision making parameters and necessary knowledge inputs were explicitly referenced by inspector fields (**Figure 4.2**). For some frequently used functionalities, such as response curves, I did create modular components which could be referenced and used by the action classes.

Because this architecture required a custom-scripted action class for each possible action, I needed extensive knowledge of the technical properties of the system to generate new AI behavior. In a professional setting with specialized programmers and designers this might be a bad solution, but for this relatively simple project with only

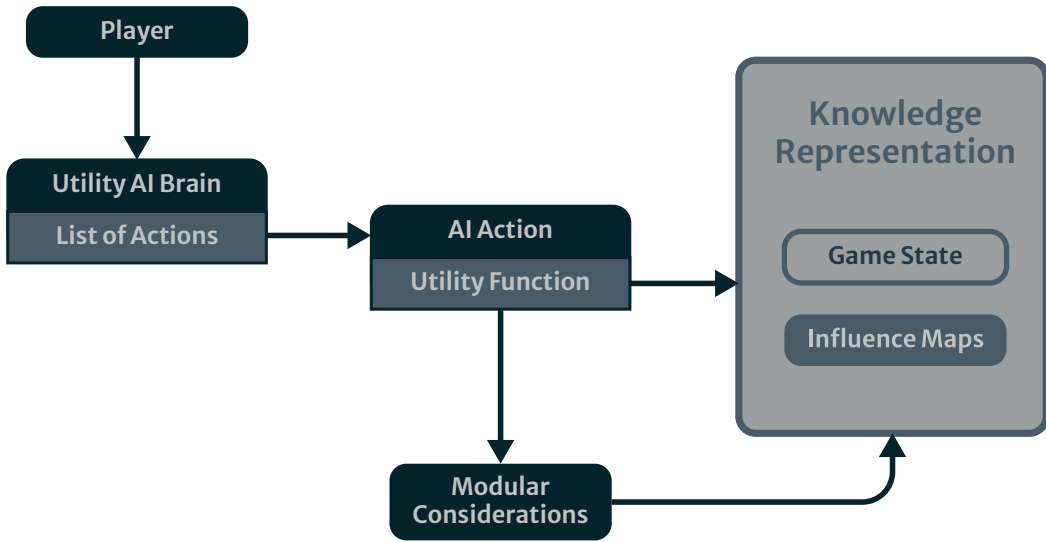


FIGURE 41
Utility AI
architecture

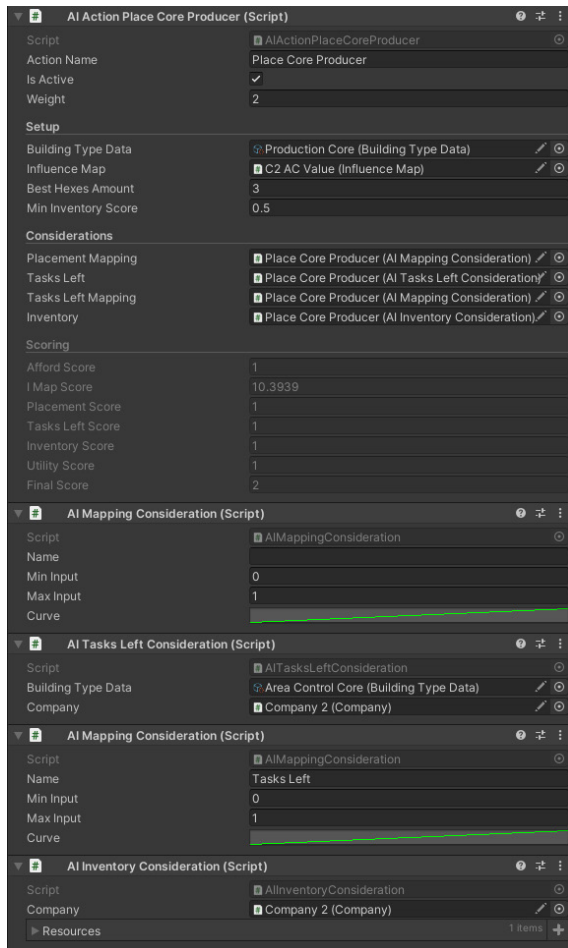


FIGURE 42
AI Action
components on a
GameObject
in Unity

one developer it worked out fine. At this stage, there were only seven unique actions to handle and I could modulate their utility functions with great design control within one code section. I also didn't have to worry about some aspects, like data passing and value normalization, which generally need to have standardized rules within more complex modular systems.

Influence Maps

Influence maps are based around the concept that objects in the game world emit influence into the world. This influence is stored inside the map, most commonly a grid, as numeric values (**Figure 43**), which can be used by the game AI to make decisions with informed spacial reasoning. The concept of influence maps was part of my game idea from the very beginning, as I envisioned an AI player that would evaluate a map for optimal building placement positions.

FIGURE 43
Influence map with floating point values

					0.05	0.09	0.12	0.13	0.12	0.14	0.14	0.12	0.13	0.12	0.09	0.05				
			0.02	0.10	0.16	0.21	0.24	0.27	0.34	0.37	0.37	0.34	0.27	0.24	0.21	0.16	0.10	0.02		
		0.02	0.12	0.20	0.27	0.33	0.39	0.49	0.56	0.60	0.60	0.56	0.49	0.39	0.33	0.27	0.20	0.12	0.02	
		0.10	0.20	0.29	0.38	0.44	0.58	0.70	0.78	0.82	0.82	0.78	0.70	0.58	0.44	0.38	0.29	0.20	0.10	
0.05	0.16	0.27	0.38	0.47	0.60	0.77	0.90	0.98	1.02	1.02	0.98	0.90	0.77	0.60	0.47	0.38	0.27	0.16	0.05	
0.09	0.21	0.33	0.44	0.55	0.77	0.93	1.08	1.16	1.20	1.20	1.16	1.08	0.93	0.77	0.55	0.44	0.33	0.21	0.09	
0.12	0.24	0.36	0.48	0.60	0.84	1.06	1.24	1.31	1.33	1.33	1.31	1.24	1.06	0.84	0.60	0.48	0.36	0.24	0.12	
0.13	0.25	0.38	0.50	0.63	0.88	1.13	1.38	1.38	1.38	1.38	1.38	1.38	1.13	0.88	0.63	0.50	0.38	0.25	0.13	
0.12	0.24	0.36	0.48	0.60	0.84	1.06	1.24	1.31	1.33	1.33	1.31	1.24	1.06	0.84	0.60	0.48	0.36	0.24	0.12	
0.09	0.21	0.33	0.44	0.55	0.74	0.93	1.08	1.16	1.20	1.20	1.16	1.08	0.93	0.74	0.55	0.44	0.33	0.21	0.09	
0.05	0.16	0.27	0.38	0.47	0.60	0.77	0.90	0.98	1.02	1.02	0.98	0.90	0.77	0.60	0.47	0.38	0.27	0.16	0.05	
		0.10	0.20	0.29	0.38	0.44	0.58	0.70	0.78	0.82	0.82	0.78	0.70	0.58	0.44	0.38	0.29	0.20	0.10	
		0.02	0.12	0.20	0.27	0.33	0.39	0.49	0.56	0.60	0.60	0.56	0.49	0.39	0.33	0.27	0.20	0.12	0.02	
		0.02	0.10	0.16	0.21	0.24	0.27	0.34	0.37	0.37	0.34	0.27	0.24	0.21	0.16	0.10	0.02			
					0.05	0.09	0.12	0.13	0.12	0.14	0.14	0.12	0.13	0.12	0.09	0.05				

The basis of my technical implementation was a generic Influence Map class that could store floating point numbers using the Hex Layer class discussed in section 2.3.1. It provided a number of useful getter and setter functionalities, but didn't actually involve any specific logic on how to populate the map with values. This task was carried out by an Influence Map Populator class. Both classes were components on a Unity gameobject in the scene hierarchy and the inspector fields allowed me to specify necessary parameters and references (**Figure 44**).

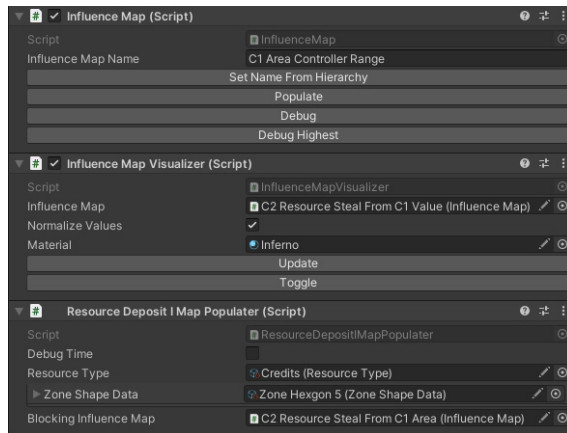


FIGURE 44
Influence map
components on
a GameObject
in Unity

There were a variety of different populator subclasses that handled the specific logic of how game state was processed to create an influence map. Some just iterated over the tiles of the map and set the values according to certain properties like area control or unblocked building space, but more advanced populaters propagated the influence of objects such as resource deposits or buildings (Figure 45). A third type of populator was used to modulate or combine the information of different influence maps and I developed classes for inverting, blurring, adding, multiplying and masking the data.

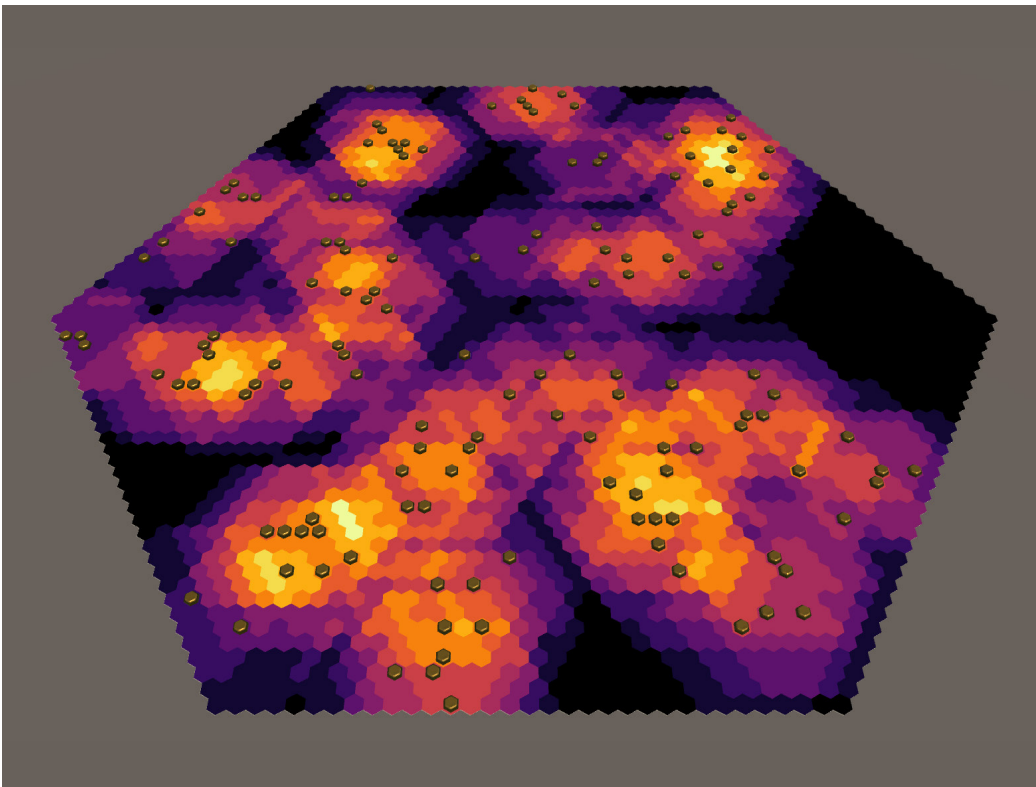


FIGURE 45
Spatial influence of
resource deposits

Because some types of influence maps, like blurring, took quite some time to compute and other types were dependent on each other, I needed to develop a quite sophisticated update process. At the top of the hierarchy was the Influence Map Manager which was activated by the Utility AI Brain and told all influence maps to populate. Maps that were dependent on other maps would wait until their dependencies were fully computed. In order to avoid freezing the update loop of the game, this process had to be fully threaded, which I accomplished with the help of the free Thread Ninja package by Ciela Spike. In the end, one whole update of all influence maps took close to a full second on my system. However, because of the turn-based game system, this had no negative impact on the gameplay experience. The AI player had to take its time for each turn anyway so the human players could keep up and comprehend what was going on.

The last crucial part of the influence map system was visualization. When working with Game AI, quality debugging is an absolute must, as designers can't hope to effectively configure an AI model without easy access to the underlying data. Fortunately, I could use the mesh generation methods I had developed earlier and extend them via simple UV mapping to show a nice heatmap. This way I was able to get a great global view of each individual influence map. For more precise evaluation purposes, I also implemented a clickable debug option that would write the numeric value of a specific tile to the console.

Action Example – Area Controller Jump

One specific action was to place an area control building inside open territory – the Area Controller Jump. The specific hard-coded utility function can be seen in **Figure 46**. At first, all scores were reset to zero. Then, the program checked if the inventory contained enough resources to afford the construction costs. If not, a utility score of zero was immediately returned.

Next up, an influence map was queried to receive its best scored tiles. The referenced influence map was a compound map type that combined data from three other influence maps (**Figure 47**). The first expressed the general value of placing an Area Controller at all free positions of the map, which was also a compound map processing even more maps that conveyed resource gathering potential and more. The other two maps were masking out the AI player's territory and the spacial influence of all existing Area Controllers in order to only consider viable open spaces.

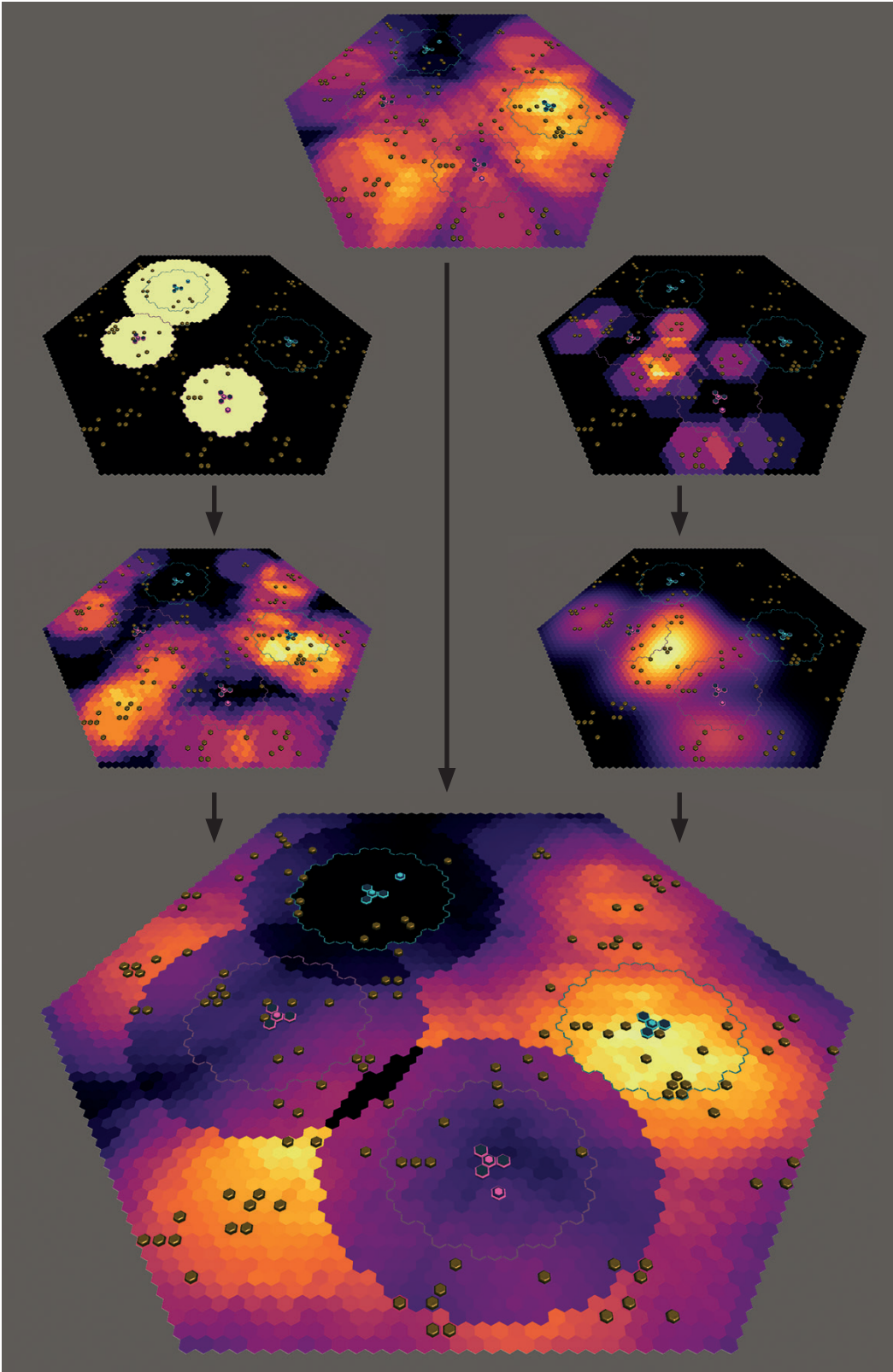
After getting a number of highly scored tiles, the program checked if the placement of an area controller was actually possible at those positions, which could be denied by blocking elements on neighboring tiles. If no viable placement position and rotation was found, the utility function would return a score of zero. Otherwise, the influence score of the valid placement position was run through a response curve, which changed the influence value, which roughly corresponded to the amount of resource deposits in the area, to a normalized utility value between 0 and 1.

```
62     protected override float CalculateUtility()  
63     {  
64         ResetScores();  
65  
66         // Check if inventory can afford construction cost  
67         _affordScore = _aiPlayer.company.inventory.CanAfford  
        (_buildingTypeData.constructionCost) ? 1f : 0f;  
68         if (_affordScore == 0f)  
69             return 0f;  
70  
71         // Get best hexes from influence map  
72         List<Hex> bestHexes = _influenceMap.GetHighestHexes  
        (_bestHexesAmount);  
73  
74         // Try to find placement  
75         if (!PlacementFinder.TryGetPlacement(_buildingTypeData,  
        _buildingTypeData.buildingPlacementType,  
76             bestHexes, _aiPlayer.company, out _placementInfo))  
77             return 0f;  
78  
79         // Map score with response curve  
80         _iMapScore = _influenceMap.GetValue(_placementInfo.anchorHex);  
81         _placementScore = _placementMapping.GetValue(_iMapScore);  
82  
83         // Get score from tasks left considetation  
84         int tasksLeft = _tasksLeft.GetValue();  
85         _tasksLeftScore = _tasksLeftMapping.GetValue(tasksLeft);  
86  
87         // Return final score  
88         return _placementScore * _tasksLeftScore;  
89     }
```

FIGURE 46
C# code of a
scripted utility
function

At last, the utility function considered the tasks left, which represented the remaining turns minus the area controllers which could still be dismantled, because dismantling area controllers would always have priority at the end of the game. This number was mapped and normalized by another response curve. With this consideration the utility function would lower the score of a possible Area Controller Jump action as the number of tasks decreased. This accounted for the fact that with just a few tasks left, it became increasingly useless to build another area controller into an open space because there was no time left to gather enough resources to get back the initial investment. In the end, the utility function returned the final score by multiplying the task left score with the placement score.

FIGURE 47
Influence map
composition



2.6.4 AI Configuration

After the architecture and most of the basic logic was implemented, I started a process of iterative configuration, which was all about playtesting, observing the AI player's decision making and then adjusting the utility functions to improve it. Similar to the game design process, I focused on the most glaring issues first and then gradually shifted towards more subtle tweaking and balancing.

As stated before, good debugging is a must when trying to effectively configure game AI. I needed to understand exactly how certain decisions were made in order to adjust them properly. Initially, I envisioned a comprehensive debugging tool, with its own Unity window that visualized the decision making process in a similar way to the staircase structure discussed in section 2.6.2. However, this would have been a lot of work, so instead, I decided to use simpler methods for now. The Utility AI Brain class logged all action scores to the console, giving a good first overview (**Figure 48**). When I needed to analyze a specific action score in more detail, I could look at the action component in the inspector, where I made sure to display all relevant subscores of the utility functions. Of course, a lot of functions heavily relied on the data from influence maps, which I could look at with my visual debugging tools described earlier.

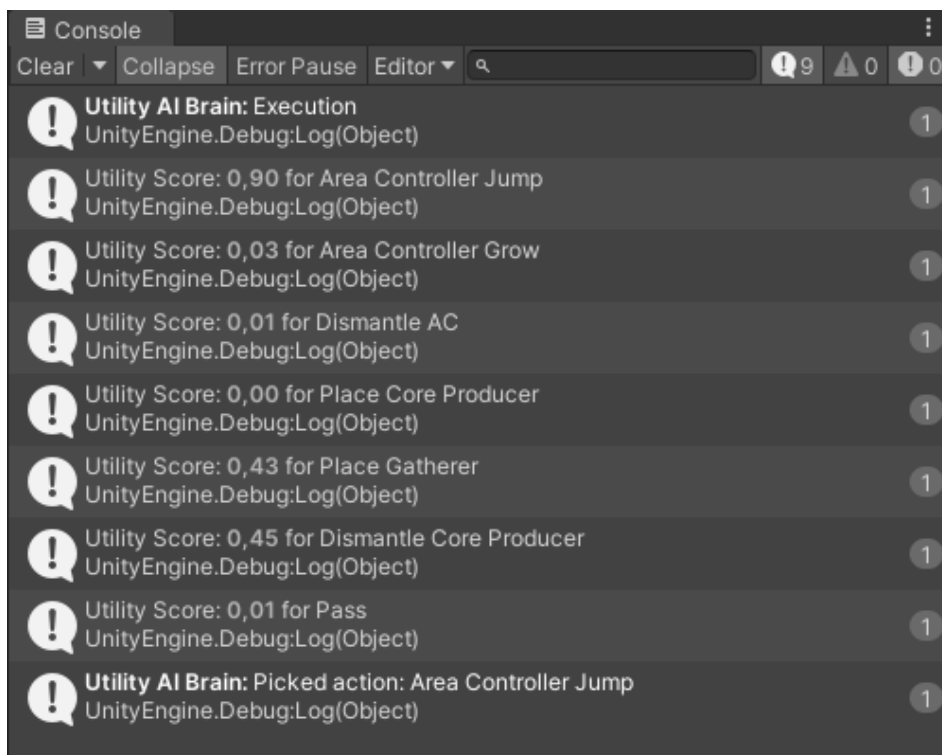


FIGURE 48
Simple Utility AI debugging in the console in Unity

With the combination of these debugging methods I was able to grasp the inner workings of the Utility AI system in an efficient manner, which enabled me to quickly iterate and configure the AI player's behavior to my liking. Once the fundamental actions were in place, the process of eradicating problematic behavior worked surprisingly well. I could analyze the scores of a specific suboptimal behavior and adjust the corresponding values to make sure the AI player would make a better decision the next time. There were some great moments early on in this behavior modulation phase, when the Game AI was generally still rudimentary, in which the AI player would surprise me with some shockingly brilliant moves. Especially the spacial reasoning ability, powered by the influence map data, was extremely competent from the beginning and while human players would ponder excessively over where the best placement position for a resource gatherer might be, the AI player made effective decisions within a fraction of a second. It didn't take very long until the AI player's competence reached the level of human players.

2.6.5 Playtesting with Human Players

Once I was satisfied with the performance of the AI player, I conducted some playtests with human players. Initially, I thought the AI player would already be too dominant, but it turned out that the playtesters were okay with a strong computer-controlled opponent. Even though most players lost to the game AI, they felt the difficulty level was a fair challenge.

Players had no problem comprehending what the AI player was doing, as it was only utilizing the common set of familiar actions and the turn-based nature of the game made sure that players weren't occupied with anything else while the AI opponent would make its move. I had also spent some time improving the presentation layer of the game since the last version. There were now more visual effects and some basic sounds that communicated what happened in the game (**Figure 49**) and as each distinct action had its own set of effects, human players had more than enough information to infer the AI player's moves effectively.

Interestingly, playtesters had almost the same relationship to the artificial player as to another human player. New players would learn the game by observing and copying their opponent's moves. They would stop and ask themselves why the AI made certain decisions and whether they should do something similar. When the AI player executed an aggressive or competent move, players would moan, complain and banter just like they did when they were playing against another human. Of course, this was the general goal and I was happy that it already worked so well – as the AI player never did any overly stupid moves, the illusion of playing an intelligent, human-like opponent was never damaged.



FIGURE 49
Visual effects for
resource gathering

2.6.6 Version 2 AI – Evaluation

The first iteration of the Game AI was a great success, as a balanced configuration of the utility functions made sure appropriate decision making was executed in almost all possible gameplay situations. The AI player was able to play the game well in all aspects and managed to emulate a very competent human player

When playtesting the game extensively, I only encountered a few fringe situations in which the game AI made a decision that could be reasonably criticized as suboptimal. One example was that the AI player sometimes built a new area controller very late in the game and didn't manage to gather enough resources to get the initial investment back. However, the few remaining cases of flawed behavior could surely be eliminated by further configuration or considering additional decision making factors. Overall, I was actually very surprised by how a few basic considerations already resulted in a well-rounded decision making process that covered almost all cases effectively. The game AI made the same decisions as human players who planned and looked ahead without having to do so itself. Many of the decision factors I listed in the decision making analysis weren't even needed. The AI didn't think about whether it needed resources for something or could afford another building after gathering a particular amount of resources. It just gathered or produced the most valuable resources at any given moment and went on from there. Ultimately, the priority staircase pattern would make sure that one action led to the next. While it was questionable if this approach would work with a more complex game system, it was sufficient for now.

The game AI played so well that it almost felt too strong when playing against human players. Especially the spacial analysis of the map was so effective that it resulted in a superhuman ability to quickly find good placement positions. Of course, playing against a superior opponent without the hope of winning is not very fun, but surprisingly most playtesters were okay with the experienced difficulty of playing against the AI player. Ultimately, it would probably be desirable to implement a mechanism to reduce the competence of the game AI. Normally, in Utility AI systems, this is done by using a weighted random selection for picking the action to execute (Graham, 2014). In this way, the AI doesn't necessarily pick the best scored action, but rather one of the best. For now, however, I was simply pleased with how well the AI was playing the game and I believed that making it less competent, if needed, wouldn't pose too much of a challenge.

Lastly, the game AI's behavior was overall a bit too consistent and lacked any interesting variety, which made the AI player feel somewhat clinical. Besides potential mechanisms for creating a variety of competence levels for the game AI, there would also be a number of approaches to create different playstyles, such as aggressive or defensive. At this stage, however, the game just didn't provide enough meaningful decisions and viable gameplay options to allow for meaningful playstyles. For now, I was content with just having one very consistent game AI and chose to not work on implementing any type of variety.

2.6.7 Workflow Evaluation and Outlook

The workflow for developing the Game AI closely resembled the game design process from earlier development stages. On my way to developing a functional AI player I went through the phases of analysis, conceptualization, prototyping and testing. The initial decision making analysis helped me to gain a deeper understanding of the game and how players make decisions when playing it. Never before have I taken such a detailed look at what goes through the mind of players playing a game I was designing and it made me much more cognisant of the subtle inner workings of the system. This understanding enabled me to define specific requirements for the AI player and formed the basis for multiple ideas of how to model the decision making process with a Utility AI system.

Developing the Utility AI architecture was very interesting but also quite a challenge. Game AI is complicated and has to be custom-tailored to fit any individual game. My research into different theoretical approaches and practical implementations didn't provide me with a suitable solution and, ultimately, I had to take the concepts I encountered as building blocks to create my own system. For the sake of time and resources, I sacrificed some modularity and abstraction and instead created scripted utility function for all gameplay actions, which were easy to handle for now. While this approach probably wouldn't scale very well, I think it was appropriate for this stage of the project.

The configuration process of the Utility AI was very interesting, fun and satisfying. It was great to see everything come together and experience the progress of the AI player in action and I was right in assuming that the handling and balancing of functions and values was something I would enjoy a lot. I was able to achieve a good behavior configuration with the help of effective debugging tools and enough adaptive iterations. However, it became clear that it's quite tricky to balance multiple utility functions against each other and I'm not sure how well my current development approach would scale with a more complex game system. A more complex decision making space would probably require a more sophisticated, modular architecture and a better, more comprehensive debugging tool.

As discussed before, the game AI could have still been improved in a lot of different ways, as there were still some edge cases to iron out, variety to add and debugging tools to improve, but I decided that this wasn't worth the effort at this point. Once again, the same old argument against AI development was made – the game design still needed to evolve and therefore it was a potential waste of time to put too much effort into developing the game AI for the current state of the game. Improvements to the AI player wouldn't change the fact that the game was still too simple to provide the quality of strategic gameplay I was aiming for. I had proven my ability to develop a competent AI player for a simple game, but now I wanted to go back to the drawing board and think about how I could improve that game in order to create a more compelling strategy gameplay experience.

2.7 Version 3 – Competitive Prototype

With the second iteration of the game, I had basically already achieved the core goal of my bachelor design project – I developed a playable strategy game with an AI opponent. Now, with just a few weeks remaining in the project’s timeframe, I needed to decide how to continue development. While Version 2 of the game was working fine, it was nothing special yet and didn’t cause any particular excitement in players. It was an interesting little prototype with potential, but nothing more and so my personal goal of creating an exciting strategy game wasn’t met yet. Therefore, I decided to fully focus on game design for now, even if it meant not implementing any more game AI. As this was probably going to be my last game design iteration, I wanted to make it count.

2.7.1 Conceptualization

Like before, I started the conceptualization phase by defining the most glaring issues of the last iteration. There were two main problems with Version 2 of the game. Firstly, the gameplay was missing strategic depth, as the game system didn’t afford enough strategic options and interesting decisions. This resulted in players feeling like they had seen it all after playing only a couple of rounds. Secondly, the game still lacked interesting area conflict gameplay. The area control mechanics were too stringent and the game didn’t provide any distinct aggressive actions. While players really wanted to engage in spacial conflict, without the right mechanics and incentives, they weren’t able to.

My initial group of ideas to improve strategic depth and spacial conflict revolved around four main areas (**Figure 50**). I planned on differentiating the area control mechanics, increasing the complexity of the economy system with more resources and production chains, adding more specialized buildings and improving the map design. Some of the ideas were already very concrete and fleshed out, while others were still quite vague and would have to be worked out in the later prototyping phases.

I knew the volume of my new ideas was very large, but I saw great potential. I could really see how this group of mechanics could come together to form a coherent whole and create an improved level of strategic gameplay. The concept was a much bigger step forward than the previous design iterations, but I had the feeling I could keep it under control and finish it in time. My plan was to implement the main ideas technically and then iterate on their design in a focused prototyping and playtesting phase. The following sections discuss the most important mechanics and their design and prototyping process in detail.

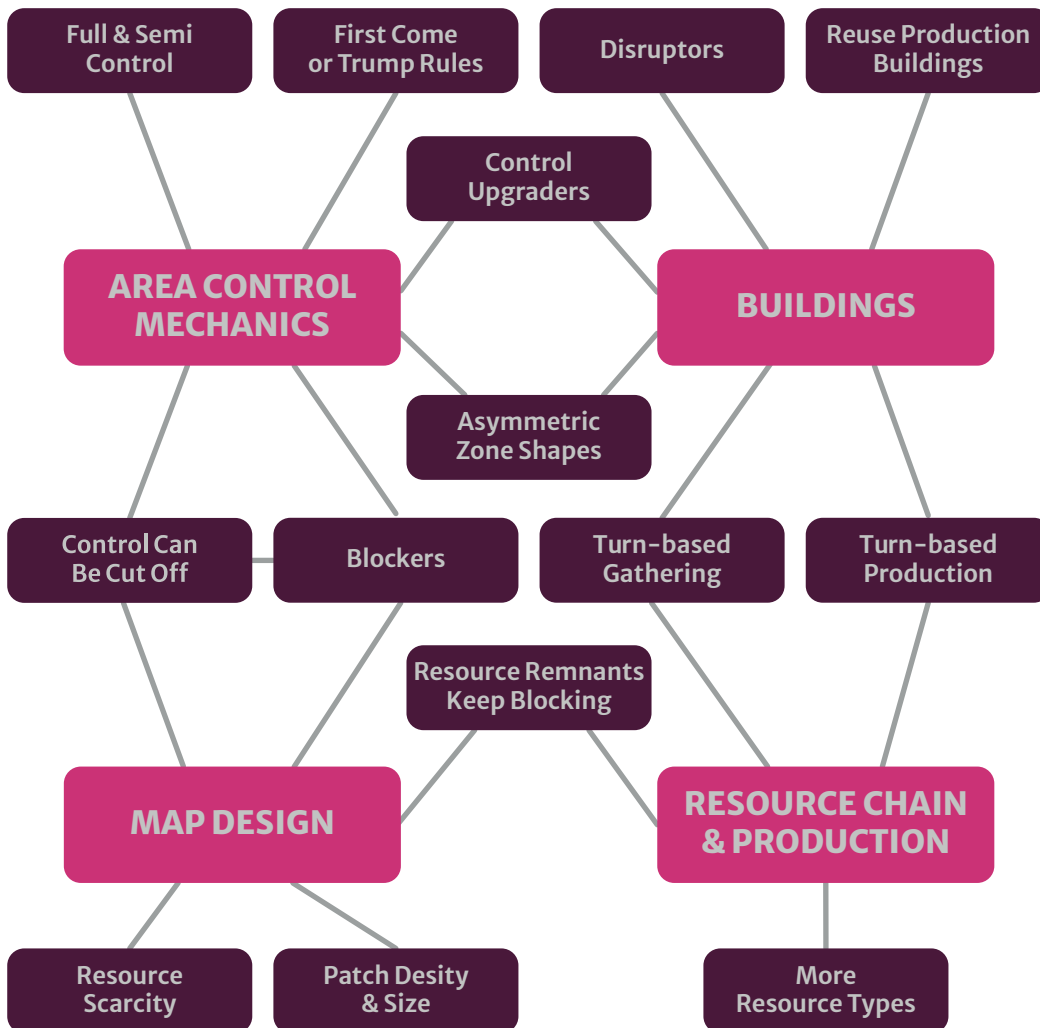


FIGURE 50
Idea mind map
for Version 3

2.7.2 Area Control

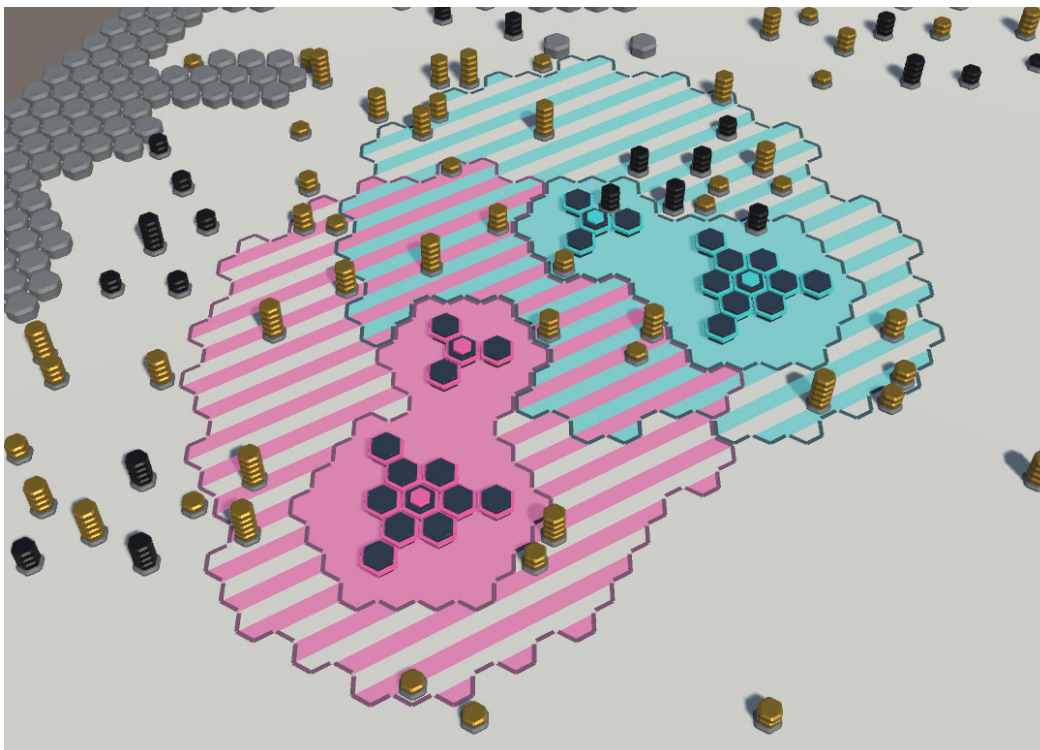
Area control was perhaps the most important focus of this design iteration. Next to the economy management, it was at least half of what made the game intriguing. I knew that in order to create more interesting spacial conflict, area control was the first mechanism that needed a significant modification.

Full and Semi Control

Up until now, area control was absolute. Once it was established there was no chance to contest it in any way. This resulted in a strong first mover advantage and disincentivized engaging into contested spaces. My solution to this issue was the introduction of full and semi control. Full control works exactly like before – only the controlling player is allowed to place buildings and gather from resource deposits. Semi control,

FIGURE 51
Full, semi and
shared control
states on the map

on the other hand, still allows players to create buildings, but not exclusively. When the semi controlled areas of two opponents overlap, both are allowed to place buildings and gather from the same resource deposits, which effectively creates a contested space in which players must fight over building space and resources (**Figure 51**).



The details of how full and semi control worked changed quite a bit throughout early prototyping. In the beginning I designed an additive first come first serve ruleset (**Figure 52**). Players could add two semi control claims to create full control. This, however, felt surprisingly awkward and wasn't even very useful in practice. Furthermore, when one player established semi-control and the opponent would add a full control claim, the result was shared control. This made sense, but was still very conservative in regards to enabling aggressive gameplay.

Ultimately, I ended up with a more aggressive ruleset. Two semi control claims of the same player would only result in semi control. This led to a higher percentage of semi controlled areas, which are generally more vulnerable to confrontation. Additionally, a full control claim would now trump all semi control claims of another player. This meant that players could actually overwrite their opponent's semi control. An important follow-up question that arose from this rule was what would happen to the buildings standing on the conquered area. For the sake of aggressive gameplay potential and logi-

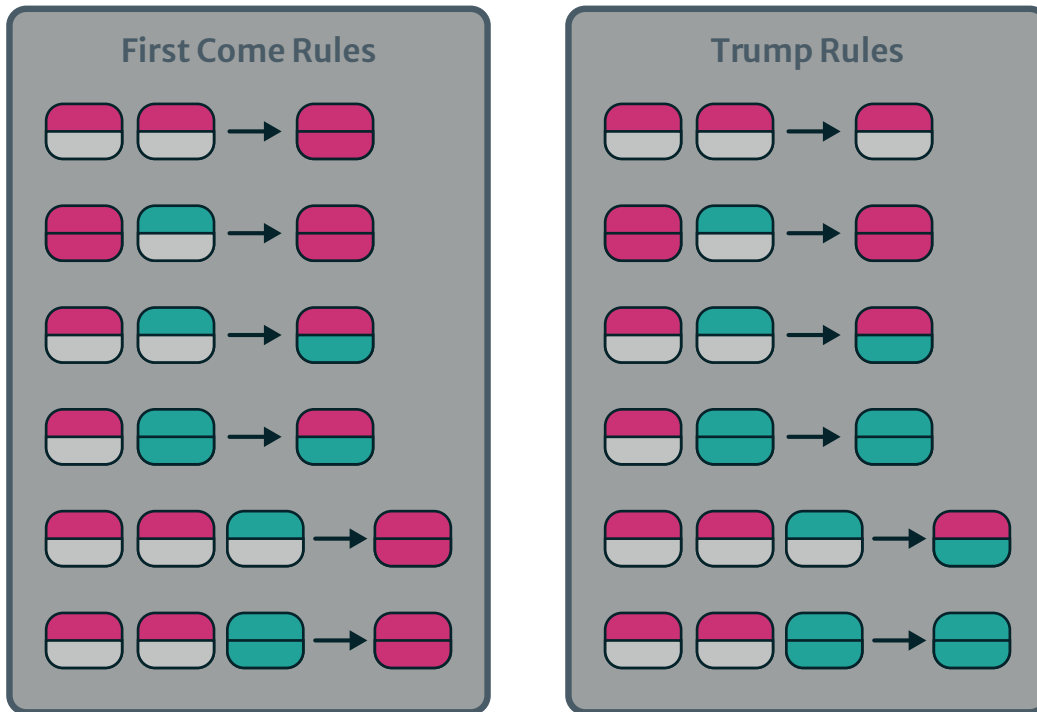


FIGURE 52
Evolution of area control rules

cal coherence, I decided they would be conquered as well. This rule effectively mitigated the first mover advantage and made conflict over controlled areas much more dynamic and interesting.

One last rule that made area conflict more exciting was that controlled areas needed to be connected to their area control buildings. In combination with the rule that full control trumps semi control this resulted in the possibility of cutting off areas (Figure 53).

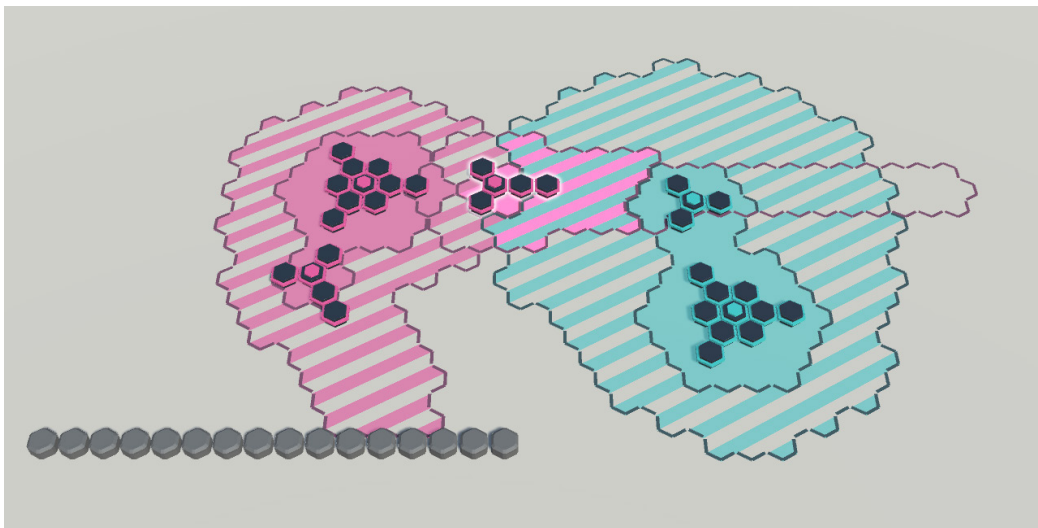


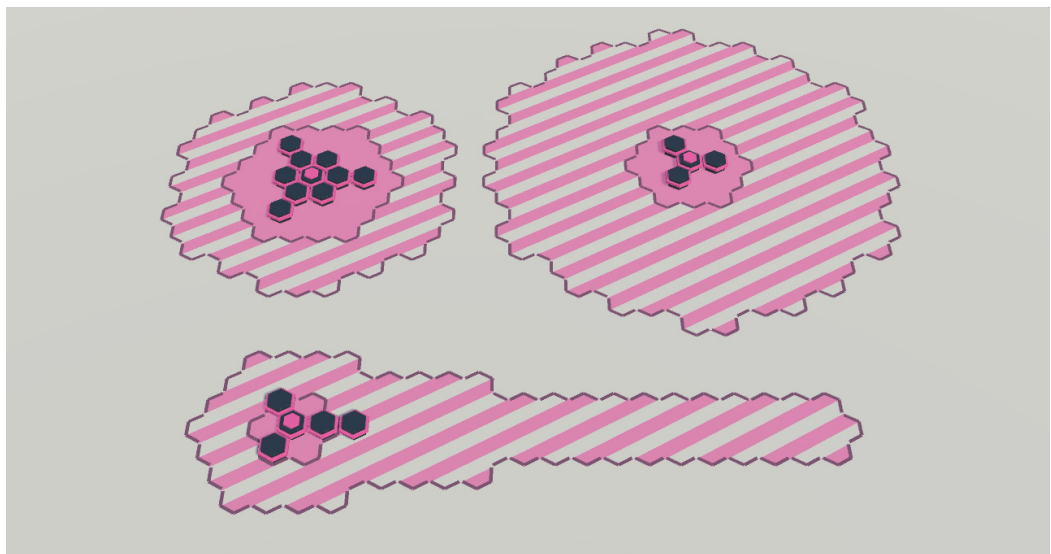
FIGURE 53
Area control cut off by full control and map blockers

Area Control Buildings

Version 2 of the game included only one area control building which could be utilized in two different ways – placing it inside or outside of controlled territory. These two different use cases and the different construction costs were confusing to many players. Like discussed before, sometimes multi-purpose mechanics aren't simplifying but rather complicating gameplay and comprehension. Therefore, I decided early on that Version 3 would have one distinct building for each of the two purposes. I called the building that could be placed everywhere Area Control Jumper and the one that must be placed inside controlled territory Area Control Grower. To make the Jumper more distinct I gave it a bigger building shape that suited its higher cost. I also used the opportunity to give the two buildings different area control ranges for full and semi control. The Jumper creates a smaller overall area, but with a bit more full control than the Grower, which, on the other hand, creates a large semi control area that is very useful for expanding existing territory.

With the new area control mechanics, it made sense to add even more area control building types. In order to add a bit more variety to expansion, I implemented the Area Control Reacher, which functions exactly like the Grower, but has an elongated, far reaching area shape. The Reacher's ability to expand over long distances in a particular direction can be very powerful, but its area shape is also vulnerable to potential cut offs by opposing players. One more special building that I developed to work with the new full and semi control mechanics was the Area Control Upgrader. This building was able to upgrade existing semi control into full control. This function made it a powerful option for defensive protection and offensive aggression alike. I could have thought of a few more area control building types with different functionalities and shapes, but I was content with the set of four. Each of them had a unique purpose and character that truly justified its own distinct building (Figure 54).

FIGURE 54
Unique area shapes
of area control
buildings



Technical Implementation

The implementation of the area control mechanics was perhaps the most difficult technical challenge of this project. Throughout the development of the third version of the game, the algorithm to compute the area control state of each tile had to be adjusted numerous times. What began as a simple 20 line method grew to be a complicated interconnected structure of multiple classes and methods consisting of nearly 400 lines of increasingly messy code. The main mechanisms that complicated the algorithm dramatically were the trumping rule, the upgrading functionality and especially the need for area connection. Before the program had to check for area connectivity, every tile could compute its state from the list of area control building claims alone, but now a new process was needed that checked the connections from a global perspective. The two processes had to be run consecutively multiple times – first check the state of each tile, then check if the areas are connected. If they aren't connected, check the state again but ignore the disconnected claims. This, however, was still not sufficient enough because it was possible for full control to be disconnected. But this disconnected full control actually cut off other control. Therefore, the whole process had to be run again for the affected tiles. An example of a complex situation with multiple cutoffs and overlaps can be seen in **Figure 55**.

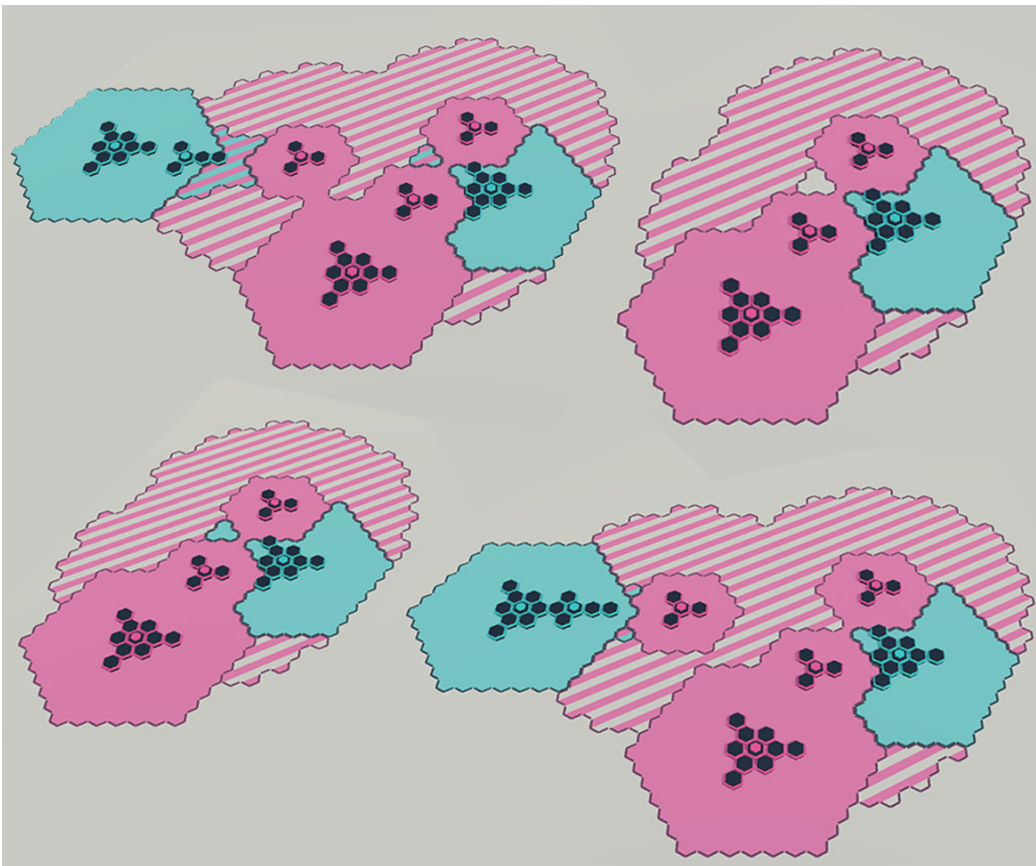
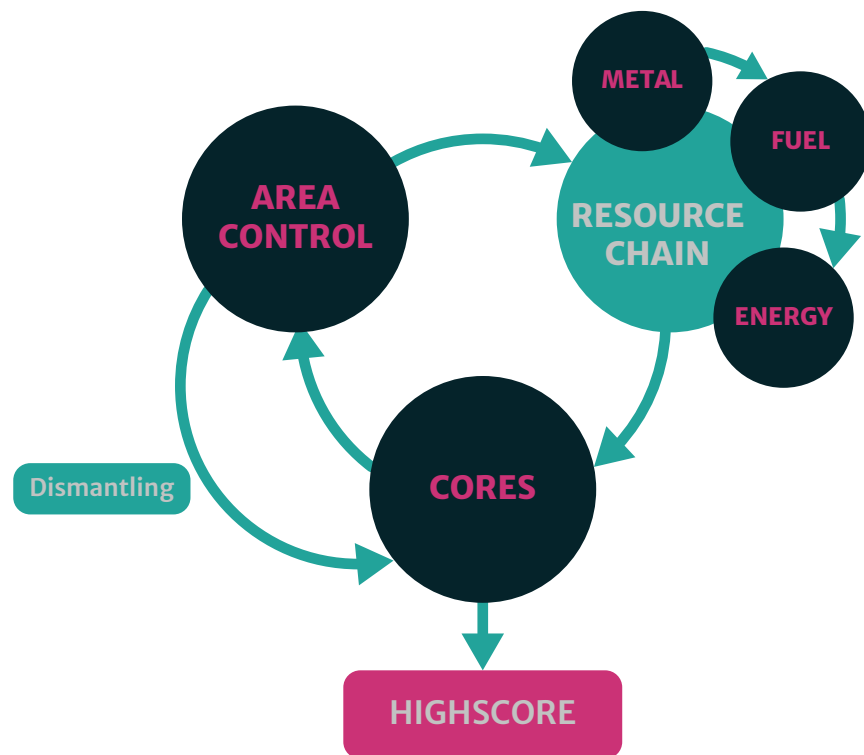


FIGURE 55
Problematic area
control situation
with disconnected
full control

2.7.3 Economy

The economy system of the second version of the game was very simple – players gathered a standard resource from the map and used it for constructing buildings and producing the special core resource, which was needed to acquire area control. This system effectively connected the two main mechanisms of the game – economy management and area control – and I wanted to keep it that way. However, I also wanted more variety in the economy system. My solution was to preserve the general core loop and only extend the standard resource into a more complex production chain (Figure 56). The goal would still be to get more cores, but the path would be more complicated.

FIGURE 56
Enhanced game
loop of Version 3



Resource Types

After some back and forth, I decided to include four resource types, each with a clear, distinct role. The first resource type was metal, which could be gathered from the map and was used for constructing buildings. The second resource type was fuel, which could also be gathered from the map. Fuel was used to produce the third resource type – energy. Energy was the only true multi-purpose resource. On one hand, it was used to produce the fourth and final resource – cores –, on the other hand, it could be

spent on a variety of buildings with special functionalities that will be discussed in later sections. The role of cores stayed exactly the same. They were still used in constructing area control buildings and had to be accumulated to win the game.

Scoring

In Version 2 of the game, the final score of players was determined by the amount of core resources in their inventory after the final turn. The counts were almost always between 4 and 6 and a lot of times both players ended up with the same score. I didn't like that games resulted in draws so often and wanted to find a way to differentiate the scoring.

The simple solution was to score not only cores, but all resource types. However, I didn't want to stop there because I saw potential to intertwine the scoring mechanism with the narrative aspect of the game. As discussed before, the game was played on a distant planet. and, initially, I had the idea of spaceships visiting the planet to trade with players. These spaceships were supposed to have limited cargo space which would have functioned as a nice control mechanism to regulate buying and selling. While trade ships wouldn't be introduced in this design iteration, I borrowed the cargo space idea to regulate the final scoring process. Players would arrive and leave the planet with a spaceship that only had a cargo space of 35. This meant they started the game with 35 resources (32 metal and 3 cores) and once the game was over, they could leave the planet with only 35 resources. The only resource type that was exempt from this rule was energy. I made this decision to solve a few systemic issues and it turned out to be an interesting exception to the rule that worked out great. Thankfully, energy was also the only resource type where this made logical sense.

So the final score of players was determined by getting points for a maximum of 35 cargo resources (cores, fuel, metal) plus points for all energy resources (**Figure 57**).



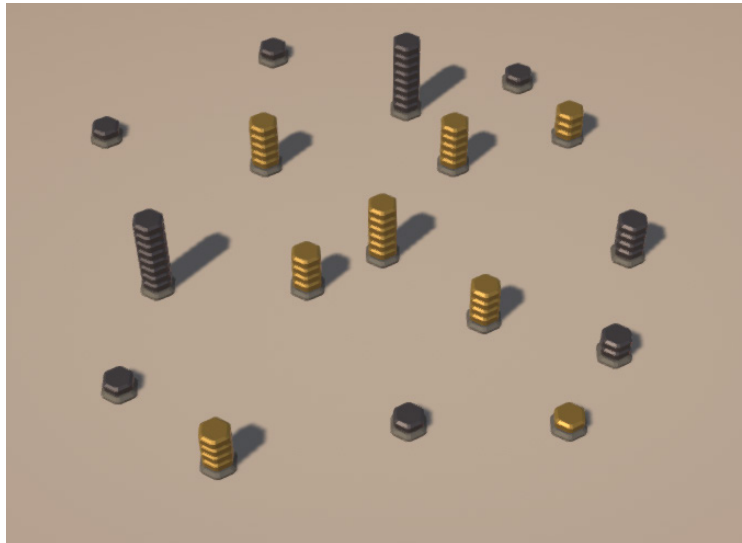
FIGURE 57
Scoreboard

As each resource type gave a different amount of points, players had to think strategically about how to manage their inventory late in the game. While this scoring system definitely enhanced the strategic depth of the game, its real value was in the narrative setup of the game. Right from the beginning of the game, when players got their starting resources and were told the win condition, they were instilled with the idea that they would have to leave the planet at the end of the game. This made it absolutely clear that the gameplay wasn't about building up a large colony and players were a lot more accepting of the dismantling mechanic.

Resource Gathering

As stated earlier, there were now two different resources that could be gathered from resource deposits on the map – metal and fuel. First I thought it would be interesting to have only one generic resource gathering building, but, in the end, I used a distinct gathering building for each resource type. Having a dedicated fuel gathering building made the decision to start gathering fuel more deliberate and strategically significant. Resource deposits were now able to contain multiple resources (**Figure 58**). Individual resources were depleted by the new periodic gathering mechanism, which will be discussed in the following section. This allowed opposing players to gather from the same resource deposits, further enhancing the possibility for competitive gameplay in contested areas.

FIGURE 58
Stacked resource
deposits



Another element I added to the game was passive gathering. Buildings which gathered passively didn't require resource deposits and continued to periodically gather resources forever. One such building was the Solar Panel building which gathered energy. Passive gatherers were supposed to provide an alternative option in the strategic space

of the game. They took very long to gather resources, but over the course of the full game, they can potentially have a higher return on investment than the standard gatherers. While the passive gathering mechanic definitely provided an interesting option to deviate from the standard strategy, I had quite a hard time integrating it into the game system, as I always feared that it could disincentivize spacial conflict. Why would players fight over patches of resource deposits if they could just build passive gatherers elsewhere?

Production

The new production chain was still only a one-dimensional sequence but now involved three instead of two steps and each production step had its own dedicated building. The Power Plant processed the base resource fuel into energy and the Fusion Factory processed energy into cores. While the metal resource wasn't involved in any production processes, it's usage in building construction still made it a necessary requirement for assembling a functioning production chain.

2.7.4 Turn Counter Mechanic

In the last version of the game, the economic functionalities – gathering and production – happened instantly when buildings were constructed. After this one-time effect the buildings were essentially obsolete. This would change in Version 3. Building functionalities would now happen after a certain amount of turns. For example, a Metal Mine would gather from metal deposits every two turns. The remaining turns were visualized by a turn counter GUI over each individual building (**Figure 59**). This mechanic made the gameplay more dynamic and interesting, as players now had a lot more processes to manage and coordinate at the same time. The turn counter also provided many attractive opportunities and advantages for the game design. I was able to come up with a number of associative mechanics and other interconnections within the game system. Additionally, the turn counter also provided useful parameters for gameplay balancing.

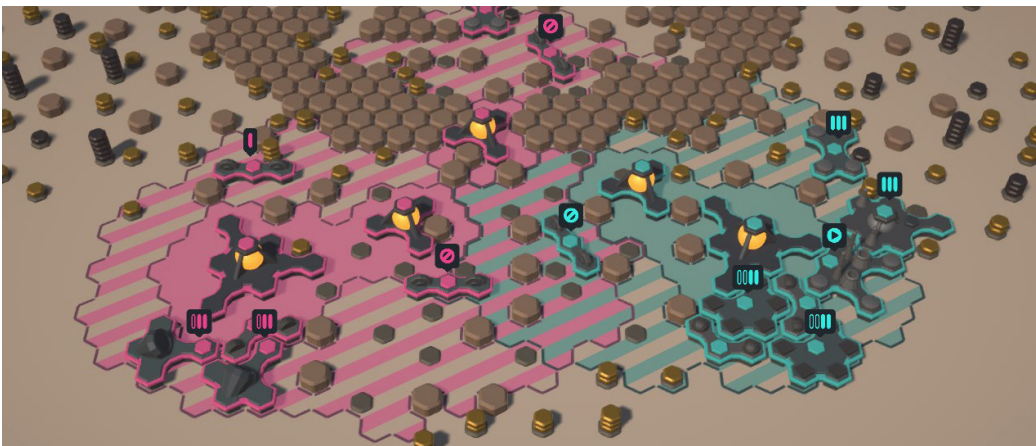


FIGURE 59
Buildings with
World GUIs
displaying turn
counters and
other states

Enhancing and Disrupting

The first mechanic to interact with the turn counters was enhancement. The Enhancer building had the effect that all counting buildings in its range would finish one turn earlier. For example, the Power Plant would take 5 turns to process fuel into energy, but with the effect of one Enhancer it would only need 4. This number could be further decreased by placing additional Enhancers, which resulted in players building compact bases to enhance as many turn counters as efficiently as possible (Figure 60). This was a great way to further emphasize the puzzle-like gameplay dynamic of building combinatorics.

FIGURE 60
Compact energy
production base



The second mechanic to affect turn counters was disruption, which was basically the opposite of enhancing. The Disruptor building added 3 additional turns to counting buildings of the opponent to slow them down. This effect could be reapplied after a cooldown period of 5 turns. The disruption mechanic was definitely the purest offensive action I came up with in my efforts to provide more potential for aggressive conflict.

Automatic Resource Distribution Issues

The issue of automatic resource distribution came up multiple times while I was designing the new economy system. One example was production. My initial concept was a production building that takes a certain amount of input resources and produces an output after a number of turns. After one production cycle, the next would automatically begin and require the input resources once again. Such a mechanism would necessitate an automatic transfer of the input resources from the player's inventory to the production building. However, this process of automatic resource distribution posed one big

question – what would happen if the inventory didn't have enough resources? What if there were two production buildings requiring new input resources but the inventory only contained sufficient funds for one of them? One natural solution would be to give resources to as many production buildings as possible, but this only raises the question of how different buildings should be prioritized. Furthermore, I would have to define what would happen to the buildings that didn't get any input resources. Would they just wait until their next chance? And what if a player wants to use energy resources for the construction of a Disruptor building but their remaining funds get automatically distributed to Fusion Factories? I was very surprised by how many issues emerged from this simple concept, especially because it is very present in many economy management games, but even after analyzing a number of different approaches from other games, I found no solution that seemed appropriate for my relatively simple, turn-based game.

Next to the production, the mechanism of building maintenance would also require automatic resource distribution. This was a concept that was really important to me, as I saw great potential for it to function as a negative feedback loop. Furthermore, maintenance costs would be a great incentive to dismantle buildings. However, the problem was the same as with production – what would happen if the maintenance cost of a building couldn't be paid?

A similar issue came up when I was thinking about including some form of storage mechanic, as I thought it would be interesting if the player's inventory had a limited capacity. Players would have to place storage buildings and think about when it was okay to dismantle them. This would provide a great strategic challenge and tie into the cargo space theme of the spaceships. I really liked this idea because it connected so well to multiple aspects of the game system, but it introduced an issue that was basically the same as the resource distribution problem, only the other way around – what would happen if the storage capacity was full? If a player gained multiple resources from gathering and production at the same time, which would be prioritized and put into the inventory?

In the end, I had to realize that any type of automatic resource distribution mechanic would create too many problems to be worth the value of the aforementioned concepts. I had to adhere to my design principle of simplicity and cut everything out. This meant no maintenance and no storage capacity. Production, however, was an integral part of the system and couldn't be discarded. My solution was a restarting mechanic. A production building that had produced its output now stayed inactive until the owning player manually restarted it. Of course, this was only possible when the player had the required input resources. I decided that this restarting action would also cost a full turn, which meant that restarting production was actually an interesting decision with an opportunity cost. Players now had to choose between restarting and other actions like constructing or dismantling buildings. Interestingly, the restarting mechanic, which was conceived as the solution to the automatic resource distribution issue, was ultimately also used for other buildings, like the Disruptor and the Area Control Upgrader.

Congruent Systemic Gameplay

The turn counter and their corresponding mechanics were subject to a lot of change throughout the design and prototyping phases. They created a lot of systemic inter-connections that needed to be configured carefully in order to create a coherent whole. There were many details I didn't think of initially. For example, can a Disruptor disrupt an Area Controller? Or what should happen to the turn counter of a production building when it is conquered by another player? Should it just go on or should it have to be restarted?

In order to arrive at a satisfying, congruent game system, I had to redesign multiple mechanics throughout the prototyping phases. In the beginning, the Enhancer made turn counters work faster instead of finishing earlier. A turn counter affected by two Enhancers would strike off three bars instead of just one each turn. This, however, was very overpowered. The Disruptor would initially put other buildings into a disruption state for a number of turns. The disrupted buildings just stopped counting for that period of time. The Area Control Upgrader worked in a similar fashion – it had its upgrading effect for a number of turns and then went inactive until restarted. The big issue with effects that were active for a number of turns was their interrelation with the enhancing mechanic. Enhancing caused turn counters to finish earlier. For gathering and production this was a positive thing because the desired effect would happen once the counter finishes, but if the desired effect would end once the turn counter finished, enhancing it to finish earlier would be a negative thing. I fixed this issue by executing the effects of the Disruptor and Area Controller at the moment of building placement. The Disruptor added bars to opposing turn counters and then ran down its own counter as a cooldown until it can be restarted. Enhancing the Disruptor now had a positive effect – the cooldown was reduced and it could be restarted earlier. The Area Control Upgrader, on the other hand, would perform its effect indefinitely. Except for when it was disrupted. Then it would lose its effect but could be restarted again.

Ultimately, the implemented changes resulted in a turn counter system that was quite coherent and easy to understand. While it was perhaps the most difficult design challenge of this iteration, it was well worth the effort, because it created such a high level of interdependence and continuity between different systems of the game.

2.7.5 Map Design

While developing the last version of the game, I created a quite sophisticated procedural map generator. Now, however, it became increasingly clear that this approach didn't afford the design control I needed to craft reasonable playspaces. With rising numbers of building types and shapes came an increased focus on the gameplay aspect of building combinatorics. As I wanted to provide more interesting building spaces for this puzzling dynamic, I started using blocking elements. Additionally, resource deposits would now also function as permanent blockers, even after being fully depleted.

Map Editor

In order to actually design playspaces, I needed a dedicated map editor that allowed me to place elements on the hex grid in a convenient fashion. I developed a system that worked in Unity's playmode and saved the data to scriptable objects. With the help of a simple ingame UI (Figure 61) I could now dynamically place, modulate and remove map elements with the mouse pointer. Additional useful features were brush sizes, symmetry functions and, of course, saving and loading. It was definitely worth it to spend a good amount of time on creating this useful editor tool because it allowed me to iterate on map design in a fast and frictionless manner.

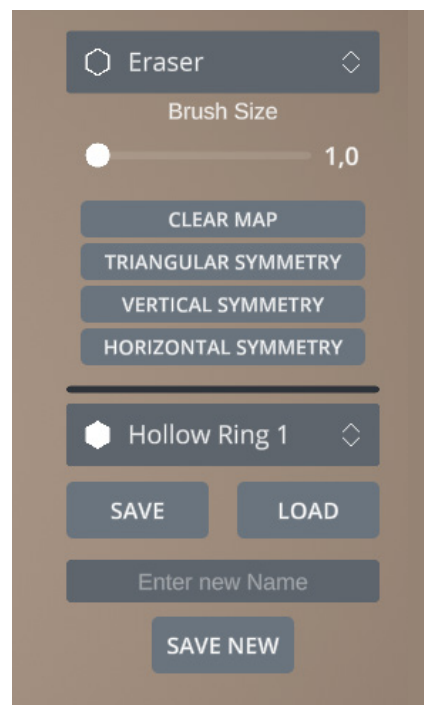


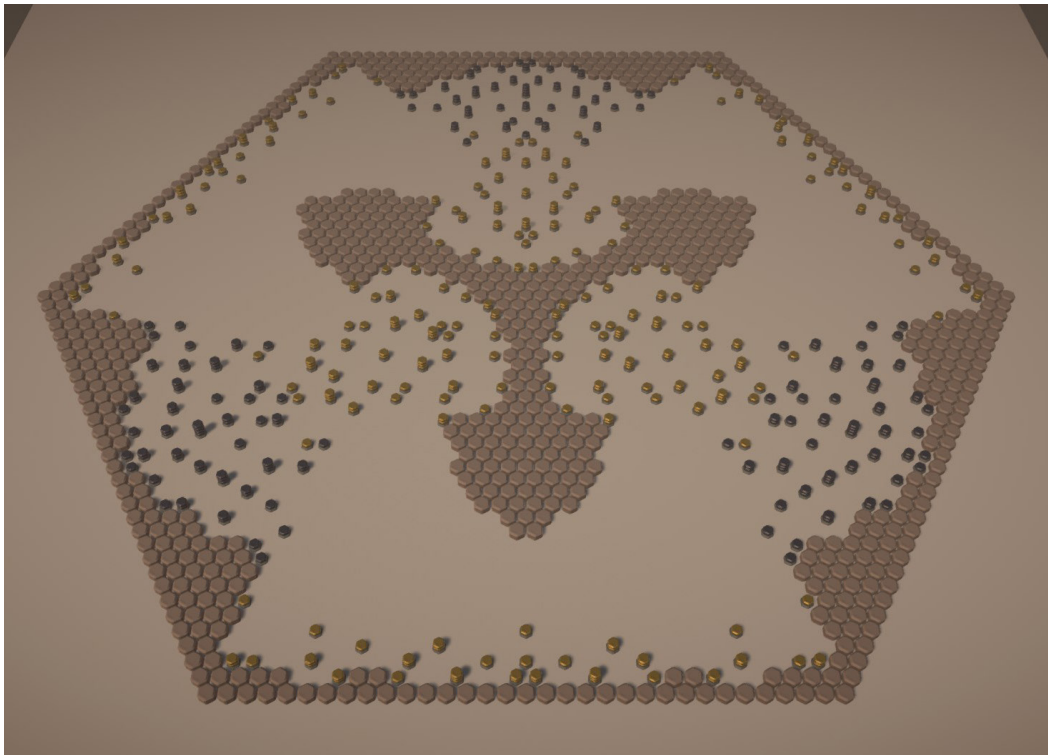
FIGURE 61
Map Editor

Map Design Process

As noted before, one of my main goals for this design iteration was to incentivize spacial conflict. Essentially, spacial conflict was conflict over resource deposits. The most basic element of map design that influenced the emergence of interesting territorial disputes was what I called the patch – any type of compact cluster of deposits. A small patch was too easily controlled by one player and it didn't make sense for opponents to compete. Very large patches, on the other hand, covered so much space that players could afford to stay away from each other. It was the medium sized patches that created the best battles over control. However, if a map had multiple patches of resource deposits, players could avoid each other again, so it became clear that the most important tool to enforce conflict over area control was resource scarcity.

One way to ensure scarcity was to keep the number of patches low. I realized that an uneven number of patches would always lead to conflict about at least one of them. For example, on a map with three patches, players would conquer one patch each and then fight over the remaining one. One archetype that came out of this idea was the ring map (Figure 62). This structure worked really well to guarantee spacial conflict because players would expand around the ring and inevitably clash. Additionally, jumping around to cut off the opponent was also a viable and exciting option.

FIGURE 62
Ring map



The generated maps in Version 2 of the game were very open and accessible, which meant that players could basically jump everywhere from the beginning. Now, I started to think about ways to enforce a specific progression for expansion. How could I make sure that players would have to start at one patch, fight over it, and only then go on to the next? One initial idea was to implement some form of fog of war to progressively uncover the map. This, however, would have required extensive work in the field of game design, technical implementation and visual communication. A simpler answer was to utilize the limited building space to control map progression. The only way to immediately jump into a certain area of the map was to build the Area Control Jumper. But what if you couldn't jump there because of a lack of space? The Area Control Jumper already had one of the largest building shapes, so by using the blocking map elements I could make sure to prevent jumping into particular parts of the map. This forced players

to jump to the fringes of these closed off spaces and then expand into them with Area Control Growers and Area Control Reachers. In one map that I called the Canyon, I took this principle to the extreme (**Figure 63**). Players had to start at the southern side of the map and then progressively expand through narrow passages to reach the resource deposits further north. This map provided many opportunities for cut offs and often played out extremely aggressively.

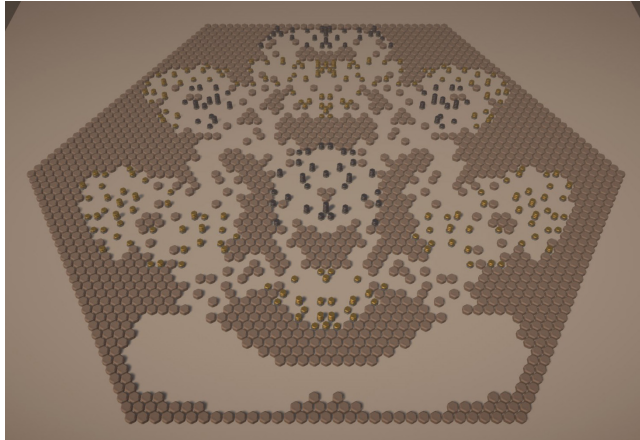


FIGURE 63
Canyon map

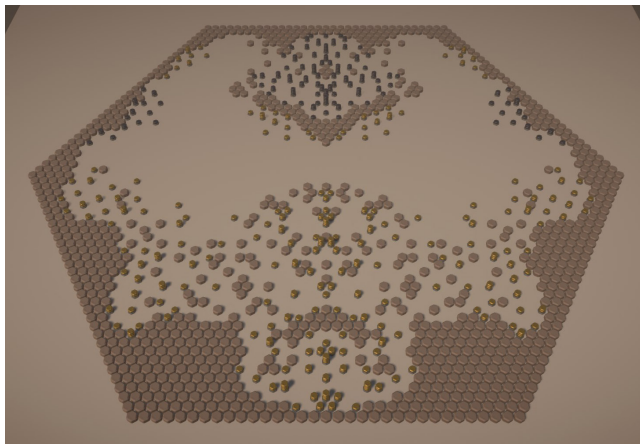


FIGURE 64
Two Sides map

I had a lot more ideas for map designs. For example, the fact that the game now included two different resource types on the map offered a lot of possibilities. One simple concept was to have the two types placed on different sides of the map (**Figure 64**), which surprisingly resulted in a very interesting gameplay dynamic. At this stage, however, I realized that it was more important to create a few quality maps than to experiment with more ideas. Each map needed to be thoroughly tested, adjusted and balanced to create an optimal gameplay experience. It was very important to me that each map

Another crucial factor for balancing the gameplay was the shape of buildings and their influence areas. For example, the influence range of an Enhancer significantly impacted its effectiveness. A larger area allowed to enhance more buildings and more enhanced buildings meant more economic productivity. The actual building shape was just as important. A large building needed a lot of free building space. Tiny buildings, on the other hand, could fit into smaller gaps and were more easily combined into compact groups. Additionally, smaller buildings were also easier to steal by opposing players, as any additional building hexes basically functioned as a safety margin to keep away rival buildings.

With the help of all these different interdependent factors it was possible to design towards any desired gameplay experience. For example, I wanted players to potentially build multiple Enhancers in one gameplay session. Therefore, one individual Enhancer couldn't be too powerful. As the Enhancer's effect of reducing turn counters by one was set in stone, I had to adjust its power by giving it a quite small influence range which meant it could only enhance a low amount of other buildings. In order to fit multiple Enhancers into a space and add to the puzzling dynamic, its building shape had to be rather small, which went hand in hand with a fairly affordable construction cost. By balancing all these parameters against each other nicely, I was able to give the Enhancer the exact gameplay role I envisioned.

In order to balance more global dynamics, I developed a way of displaying detailed economic statistics. The game tracked important economic aspects and the data was exported into spreadsheet software. This useful tool allowed me to balance the long-term effects of different buildings and strategies. One example was the passive energy strategy. A player could decide to completely ignore the fuel resources and use only passive gatherers to produce energy. When the passive gatherers were placed in a compact area with multiple Enhancers this was supposed to be a viable strategic option. With the help of my statistic tool I was able to precisely analyze and balance this strategy against the standard game plan of gathering fuel to produce energy (Figure 66). In the begin-



FIGURE 66
Economy analysis

ning, the passive energy strategy was completely overpowered and produced 50% more energy than the standard gameplay. This led me to reduce the power of passive energy gatherers by increasing construction costs, enlarging building shapes and decreasing outputs. After this first balance pass the strategy became too weak, it now produced 25% less energy than the standard. In the end, after multiple passes, I configured the passive energy strategy to be a bit more powerful than the standard. This accounted for the fact that the compact bases needed for this strategy were very vulnerable to aggressive gameplay mechanics such as stealing and disrupting, which meant that the passive energy strategy was ultimately designed to be a viable but highly risky game plan.

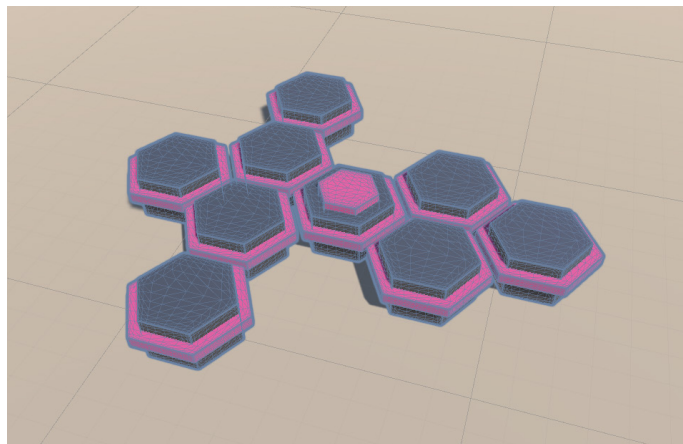
2.7.7 Visual Development

As discussed in section 2.5.2, quality visual communication is absolutely crucial for a game with discrete mechanics. Throughout the development of Version 3, I had to invest large amounts of time and effort into creating effective visual systems to make the game understandable for players, which presented a difficult challenge both in terms of design and technical implementation. Again, visual communication and technical implementation are not the focus of this paper. Nevertheless, this section will give a quick overview of the most important visual system I implemented for the third version of the game.

Building Models

Throughout the prototyping phase, I used procedurally generated building models (**Figure 67**), which allowed me to easily change the shape of buildings without any additional work. However, as more and more building types were implemented into the game, it became increasingly difficult to distinguish separate buildings on the map. Once I had decided the final shapes of some buildings, I started experimenting with individual building models to increase readability, but while trying out different approaches, I quickly realized that any model that deviated too much from just a simple flat

FIGURE 67
Procedural building
model



shape actually lost a lot of readability. From the perspective of gameplay, the buildings were basically puzzle pieces and their most important characteristic was their top down silhouette. So everytime I added something to the building model, I diminished the readability of that shape and therefore negatively impacted the gameplay experience. Ultimately, I just added very subtle elements to the building models. One exception was a large sphere on every building that was constructed with a core resource, which made sure players always had a good overview of how many cores they could still get back by dismantling these buildings (Figure 68).

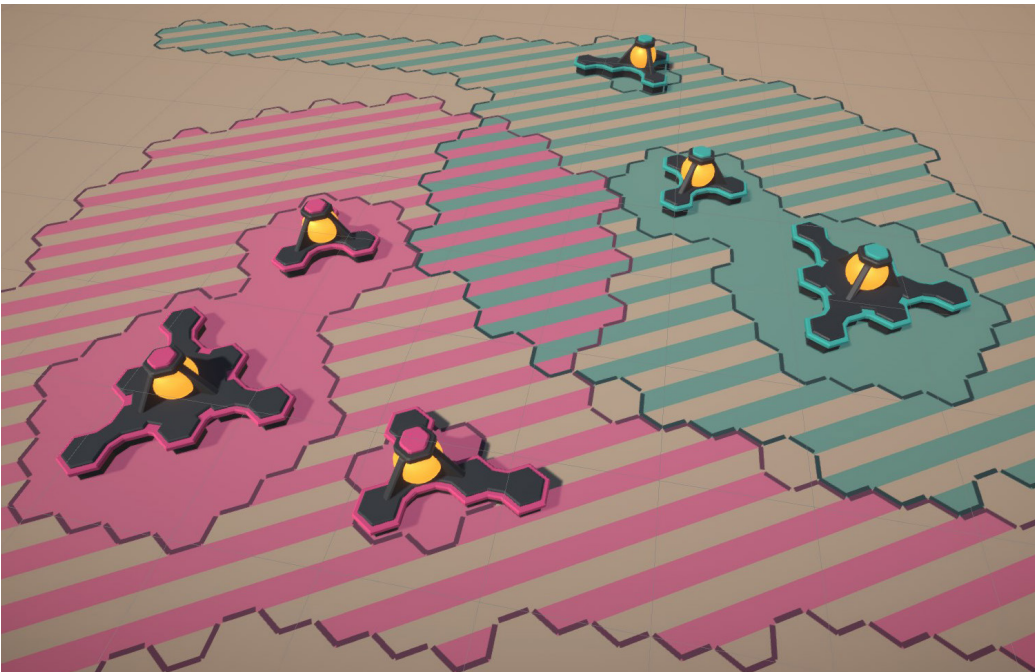


FIGURE 68
Golden sphere
indicates buildings
with built-in
resource

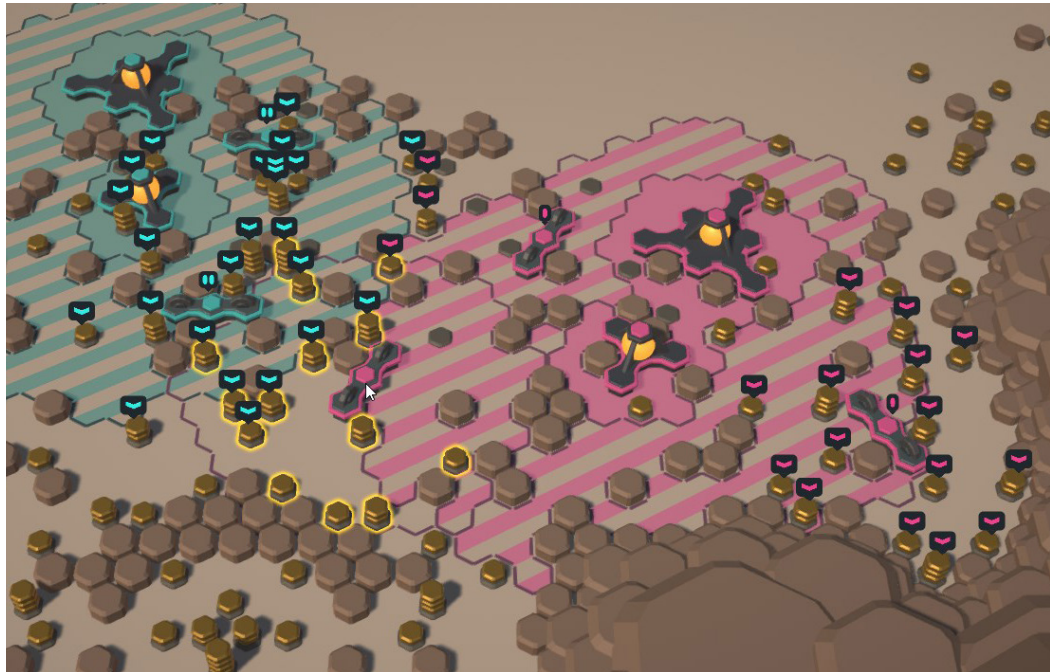
Another issue I wrestled with was the readability of the area control state where buildings were placed. The building models were effectively blocking the view of the area control mesh on the ground level. When players wanted to know if a particular building was placed on full or semi control they would basically have to look under the anchor pin of the building. The same blocking issue occurred for the border meshes for building ranges. In the end, I wasn't able to find a good solution for this issue yet. However, in practice it wasn't too much of a problem, so I was okay with leaving it like that for now.

World GUI

Now that buildings had a lot more functionalities, there was the need to visually communicate their current state. I implemented a world GUI that displayed information about the state of turn counters, the ability to restart and even obsolescence, when a gatherer had no more resource deposits to gather from. A similar GUI was

added to resource deposits and visualized the number of resource gatherers that were gathering from it (Figure 69). This information, however, was only needed when players were placing new resource gatherers and therefore it was only displayed in this context, which helped to keep the standard amount of visual information on the screen to a necessary minimum.

FIGURE 69
Resource deposit
World GUI



Context Visualization

Now that the game system included a lot more interconnections between buildings, it needed a lot of sophisticated context visualization. For example, while placing an Enhancer, the game now highlights the buildings that would be enhanced. This highlighting was mostly done with color-coded outlines.

Generally there were three distinct interaction modes for buildings – placement mode, selection mode and dismantling mode. For many building types all three modes actually visualized the same information. Others, however, like the Area Control buildings, had different context visualizations for the three interaction modes. Firstly, while placing an area control building, the area that would be added was specifically highlighted. Secondly, when an Area Control building was selected, the visualization would highlight the area and buildings that were exclusively controlled by it (Figure 70). Finally, when the mouse cursor was hovering over the dismantling button, the game visualized the state that would occur after the dismantling, which included the loss of territory and potentially an indication of a change of ownership for a building.

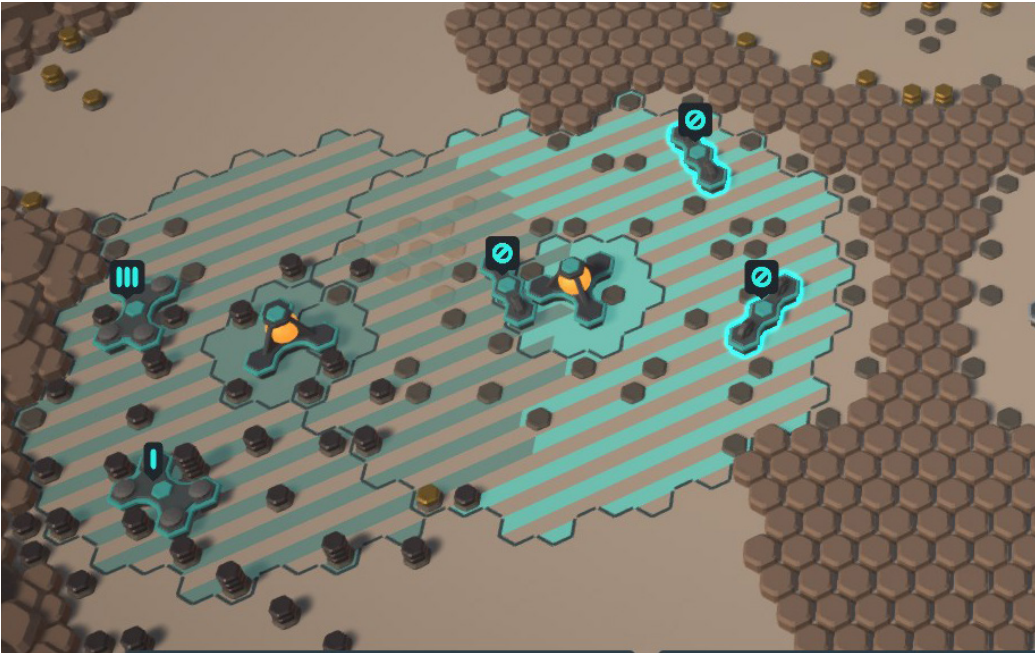


FIGURE 70
Selected area con-
troller visualizing
exclusively
controlled
buildings

Interface

At the end of the development phase of Version 3, I did an extensive update of the game's head-up-display (**Figure 71**). At this point, I had a good understanding of what information actually needed to be displayed. I thought about giving the graphics a sci-fi touch but ultimately kept the simple flat style for now because I didn't have much time left. All GUI elements were kept in a dark value to contrast the brighter game world. I



FIGURE 71
Final HUD design

adjusted the layout of the inventory panel to have a better centered view of the playspace. All building buttons were also aligned to the center of the screen and now indicated whether they were currently affordable by color-coding. I tried to simplify information by utilizing pictographic icons rather than text because once icons are understood, they are generally easier and quicker to read. However, it was quite a struggle to structure the information inside the windows effectively and there was still a lot of room for improvement, but, overall, the new interface was definitely a good step forward, both in terms of looks and usability.

2.7.8 Playtesting

In the later stage of the development of this design iteration, I heavily relied on playtesting the game with other human players. The increasing extent of competitive actions and strategic possibilities made playing the game against myself less valuable. I just wasn't able to pull off any meaningful strategies or even surprise attacks against myself.

Tutorial

The game had gotten so complex that a starting screen couldn't explain it anymore. Ideally the game would be taught to players in an extensive tutorial that progressively introduces gameplay elements one after the other, but such a tutorial would have probably taken weeks to develop, which wasn't feasible right now. Therefore, I decided to explain the game myself. For about fifteen minutes I introduced the narrative, gameplay goal, resource types and all buildings in a reasonable sequence. I actually placed all the buildings on the map to show how they functioned and interacted. This method worked great and the playtesters generally understood the gameplay rules quite quickly. My explanation was usually followed by a few questions and experimental actions of the testers and then we could start a first real game. For most players this introduction was enough to have a good idea of how to strategically approach the game and only some of the more advanced concepts, like area control upgrading, passive gathering and stealing, had to be sometimes reiterated later in the game sessions.

Timeframes

I configured the game to take 50 turns for each player, which provided the right amount of actions to progress nicely through all gameplay phases. Of course, the overall timeframe of a turn-based game is determined by how long players take for each turn. Right now, the general gameplay experience was very chess-like and players tended to think deeply about every move and its possible consequences. For most playtests the full game session took about an hour, but some testers approached the game more seriously, which resulted in game sessions approaching up to three hours.

Timer

In order to give an option to play the game faster, I implemented a timer. I used the simple delay type which gave players a certain amount of time for each turn plus a time bank. Once the delay and time bank ran out a player would automatically pass their turn. The first thing I had to realize was that the timer was not suitable for new players. In a normal playtesting setting, new players needed time to learn the game, ask questions and give feedback and the timer completely derailed this process. For more advanced players, however, the timer was not only a way to decrease playtime, it also made the game more challenging and exciting. The time pressure forced players to make decisions quickly, which led to mistakes, which made the game less predictable and generated more interesting situations and events. The resulting gameplay experience was more dynamic and less strenuous than the standard. While the time limit would definitely put a lot of pressure on players, it could actually lift another heavy burden – the feeling that one must take the time to make the perfect decision.

Playtesters

Because of the increased complexity of the game system, its mechanics and strategic options, the playtesters had a lot to learn and made many mistakes. It took quite a while to get used to thirteen different building types and their interactions. Similar to the playtests of earlier versions, players would try to imitate my playstyle because they thought I would know what to do and followed along. Sometimes this was frustrating, as I wanted to see if players would use the more advanced buildings, but they would only do so if I did it first.

It quickly became obvious that the game had gotten too complex to be playtested in just one session. One playthrough was barely enough to get an idea of the game and its mechanics. In order to playtest the more advanced strategic gameplay space, I needed more advanced players. It was actually more helpful to have one tester play the game multiple times than multiple testers play just once. This circumstance was very new to me and it turned out to be quite challenging to find the right playtesters and coordinate these in-depth playtests.

An interesting observation was that the gameplay experience of playtesters now seemed to heavily rely on player types and preferences. Some players really didn't like the fact that the game forced them to think about so many interconnected elements and possibilities, while others seemed to love it. I could definitely get an idea of why complex strategy games are more of a fringe genre.

2.7.9 Version 3 - Evaluation

Version 3 was a great improvement from its predecessor. Even though the game was now a lot more complex, it was still very coherent. All the individual parts worked very well together to form a coherent whole. The overarching narrative and the precisely defined gameplay goal managed to frame the intended gameplay more clearly and helped to contextualize the dismantling mechanic. Furthermore, the four resource types had clear, distinct roles and enabled a more structured gameplay progression. The new turn counter mechanic integrated nicely into the system and turned out to be quite intuitive. It made the gameplay more dynamic and fine-grained. With the increased number of building types and shapes emerged a more sophisticated puzzling dynamic that was enjoyed by many players. Overall, the game now provided much more interesting decisions, viable options and strategic depth. The gameplay was more competitive and less predictable. Players had more fun and were eager to play more games. I could finally say that I had a true functioning strategy game on my hands.

Strategic Depth

As noted, the strategic depth improved a lot with this design iteration. There were now more actions, more buildings, more resources, more options. Many gameplay situations provided a multitude of viable options for players. Some cases were so complicated and unpredictable that it was impossible to determine the perfect action to take. However, the game system didn't just afford interesting decisions for each individual move. Now, there were also global, long-term strategies. There were multiple different game plans like the standard strategy, passive gathering strategy, metal monopoly, heavy aggression and many more. And all of these strategies were mostly well balanced against each other. Nevertheless, I can safely say that, even after many hours of gameplay, I haven't fully explored the strategic possibility space of the game so far.

New Building Mechanics

The turn counter mechanic was a great addition to the game. It created more sophisticated, turn-dependent processes that players had to coordinate and made a lot of interesting interactions possible. The two buildings that affect the turn counter – the Enhancer and the Disruptor – provided players with more tactical options. For example, players could decide to either build two production buildings or just one that is sufficiently enhanced. Two production buildings would cost more metal while the Enhancers required more energy. The energy resource itself was perhaps the most interesting resource of the game. On one hand, it was required to produce the incredibly important core resource. On the other hand, it was also needed for all special buildings, like Enhancers, Disruptors and the Area Control Upgrader. This multi-purpose design made the spending of energy an interesting and tricky decision.

The restarting mechanic was an idea born out of a difficult problem (the automatic resource distribution issue) that turned out to be a great addition to the game. Initially,

I anticipated it to be boring and I thought players would hate that they had to manually restart their production buildings. However, it actually provided another interesting decision. Players now had to decide between restarting a building and all the other useful actions. Restarting production played a crucial role in economy management and often it wasn't obvious whether to prioritize resource gathering or production at any given moment. Restarting also played a great role outside production, as both Disruptors and Area Control Upgraders could be restarted, which gave additional tactical depth to the overall gameplay. Furthermore, a lot of times, players didn't have the sufficient resources to restart a building, which created a great immediate goal – attain the resources to afford the restart. Interestingly, another positive impact was that the restarting action, if possible, was often an easy go-to action. Players would choose to restart an important building without much thinking time. This also improved the speed and flow of late game significantly.

Area Conflict

The introduction of full and semi control resulted in a lot more possibilities for interesting area conflict. The different control states – no control, semi control, shared control and full control – provided multiple steps of escalation that were the basis for compelling territorial disputes. Especially the race for valuable resource patches would often play out in an interesting sequence of back and forth actions (**Figure 72**). Players could utilize full control to control building space and resource deposits, cut off opposing territory and even steal rival buildings.

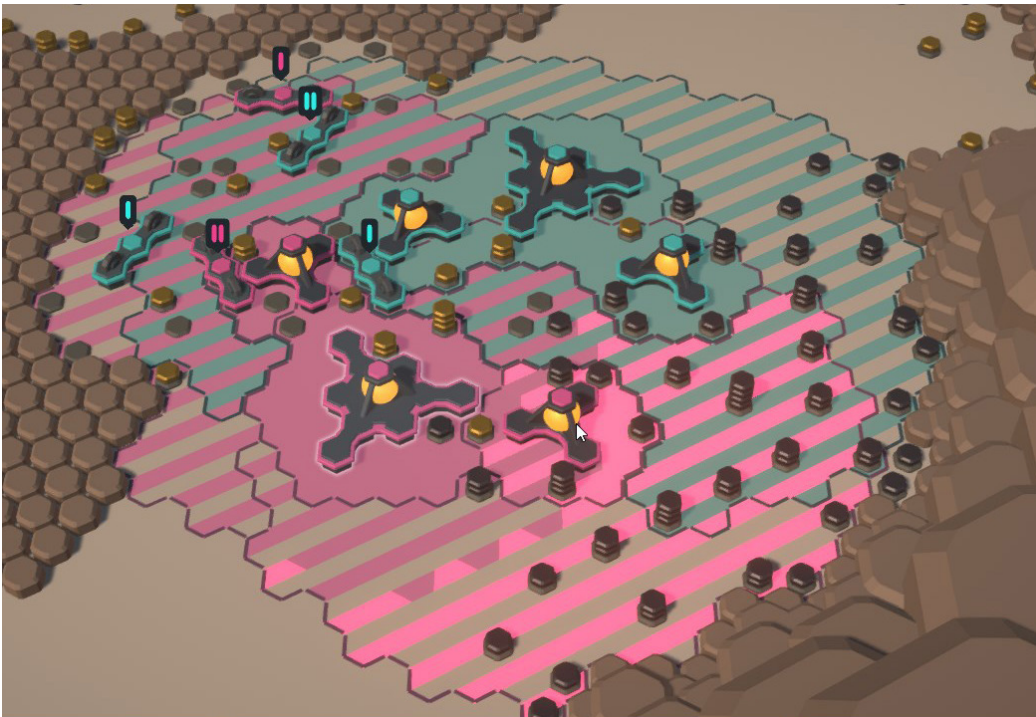


FIGURE 72
Area conflict

Players very much appreciated the new aggressive moves and always looked for openings to attack their opponent. Stealing a building was very powerful, as it was simultaneously a loss to one player and a gain to the other. Similarly, a well placed Disruptor had the potential to wreak havoc on the opponent's economic buildings and cause a massive setback. These aggressive moves could severely swing the power dynamic of a game session. Of course, this potential was highly exciting to players. Successfully pulled off, offensive moves were definitely the highlight of any playthrough.

Naturally, the potential for highly effective offensive moves made players very cautious. After being heavily punished by losing an important production building to an unexpected Area Control Jumper from the opponent, every player would start to think more carefully about how to protect their buildings. Players could place their buildings on full control or try to limit the nearby building space to prevent any stealing attempts. One advanced defensive tactic was to place an inexpensive building, like a Metal Mine, to actually block building space for the opponent.

This battle between offensive and defensive tactics became the most exciting gameplay dynamic of the game. The high potential for aggressive moves also encouraged players that were behind to continue playing on. While the game system still didn't have any true negative feedback loops, the potential for powerful offensive moves always kept the hope for a comeback alive. Overall, offensive and defensive playstyles were well balanced, as aggressive moves could be very punishing to opponents, but also required large resource investments. Defensive play, on the other hand, normally didn't require any special expenses, but players needed to take great care to not leave any vulnerable positions open to attack.

Gameplay Phases

Version 3 of the game had quite distinct gameplay phases. The early game was defined by a fresh new map and players choosing their starting positions, taking control of deposits and gathering their first basic resources. The mid game usually began when players needed to construct their first longer lasting buildings, like passive gatherers or the Power Plant. These important buildings needed to be in use until the end of the game, possibly enhanced and well protected. Therefore, clever players wouldn't just place them anywhere, but carefully pick positions that could be developed into a well functioning, compact base with minimal risk of opposing intervention. The late game was generally initiated by a low number of remaining turns and players would start to plan out their last set of actions, try to squeeze out a few more production cycles and, of course, dismantle their most valuable buildings.

Interestingly, these gameplay phases emerged naturally from the underlying game system, so I didn't need to implement any special milestone mechanics to structure the game arc. Each phase organically focused on a different mechanism of the game – resource gathering, production and dismantling. A number of distinct steps in the progression served as implicit milestones, like constructing the first Power Plant or

producing the first core resource, which gave meaning and purpose to each stage of gameplay. The only weak point of this gameplay arc was somewhere between the mid and late game. There was often a vague stretch of time in which it wasn't viable for players to keep expanding their economy, but it wasn't yet time to start the final dismantling process either. If players weren't engaged in any sort of active conflict at this point, they would lack meaningful short-term goals, which resulted in a noticeable dip of enthusiasm. Of course, a gameplay progression that has to transition from expansion to contraction can generate an awkward moment of alternation. Perhaps this is where an explicit milestone mechanic could provide some helpful structure.

I was very satisfied with how the dismantling mechanic was able to become a natural element of the gameplay. Very early on, areas of resource deposits got depleted and players were incentivized to move on (Figure 73). As they didn't have the capacity to produce the limited core resource yet, they had to inevitably start dismantling their first Area Controllers. From this moment on, the dismantling mechanic stayed relevant throughout the whole game and became the main focus at the end.



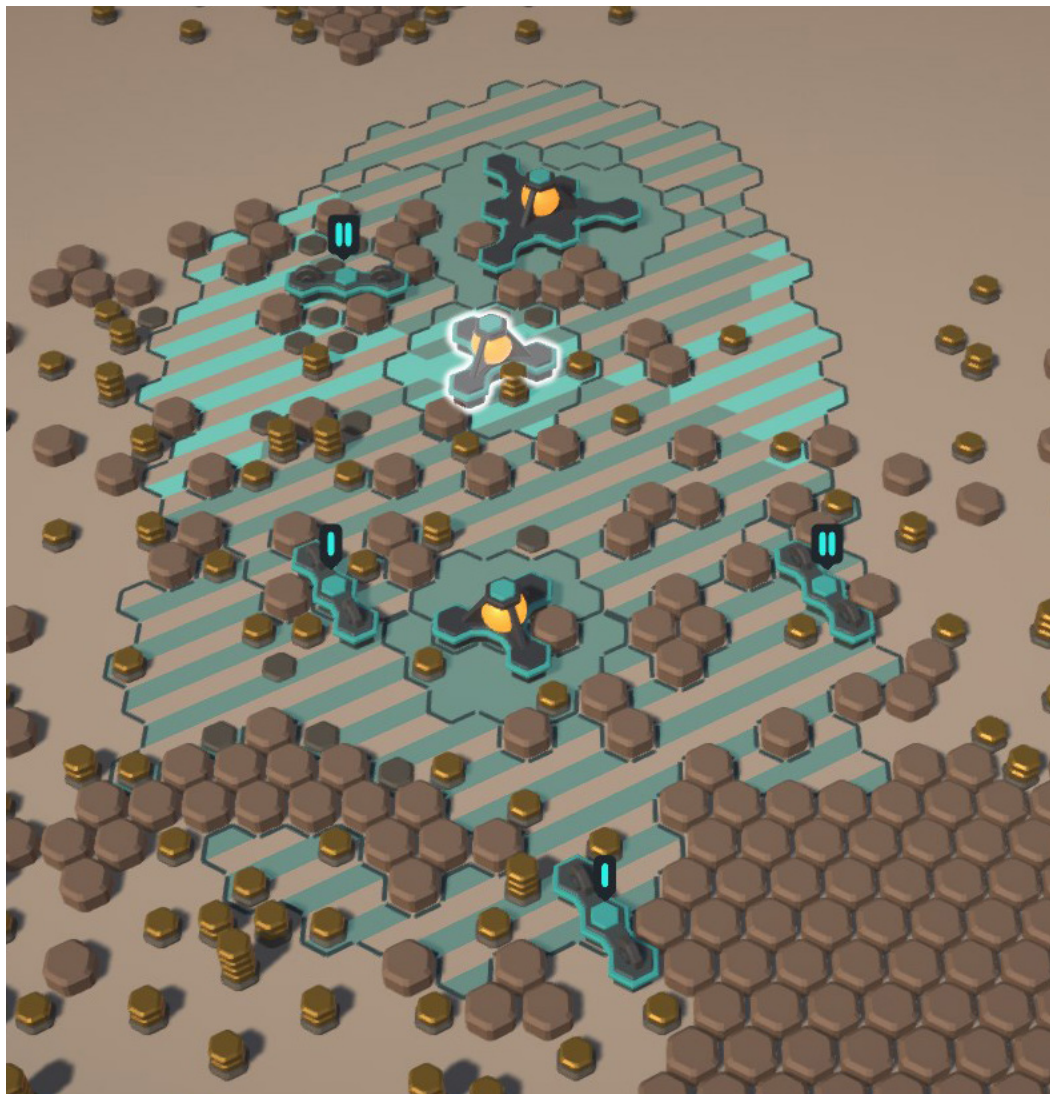
FIGURE 73
Early dismantling
of area controller

The management of core resources and area control might have been the most important aspect of strategic progression in the game. There were a number of advanced tactics players could use to deconstruct existing area control to enable new expansion more efficiently. This included the aforementioned base building, depletion of resources

deposits in one specific area before the next, prioritizing dismantling Area Control Jumpers over Area Control Growers, as they return more metal resources, and expanding in a row and dismantling the obsolete inner buildings (Figure 74).

Of course, the importance of core resources for area expansion made the first core production a major milestone. With an additional core, players had a clear advantage in seizing new territories to gather more basic resources, develop and protect their advanced economy and attack the positions of their opponent. Near the late mid game, however, expanding into new territories became less important because resource deposits became less abundant and production chains were already sufficiently developed. At this point, additionally produced cores didn't have any real impact on the remaining gameplay. They would only be useful for the final scoring. This was a bit disappointing to me and should be addressed in a potential next game design iteration.

FIGURE 74
Dismantling an inner area controller results in no building loss



End Game

Overall, the new end game was a lot more interesting than before. There were more dynamics to coordinate and actions to choose from. Players needed to make decisions about gathering additional resources, restarting production buildings and dismantling area controllers. The interconnected dependencies of area control, economic processes and the turn counter mechanic made picking the right sequence of moves a complicated and interesting challenge.

The new scoring process also had a positive effect on the end of the game. The fact that players could only get score points for 35 resources made them think deeply about their inventory management. Just a few more gathered resources or another finished production cycle could result in enough points to win the game. Players also really liked the final scoring screen after the game. It provided a nice overview of the accumulated scores and gave great feedback on the players' performances.

While the end game was generally very much improved, there was one clear negative dynamic. It was possible that the last few actions of a player were meaningless because there was no option to optimize the final score. This was mainly due to the fact that players would dismantle a lot of valuable buildings, which resulted in an overflow of metal resources which exceeded the score limit (Figure 75). In this case, dismantling another building would only result in more metal resources which wouldn't increase the score. It could be argued that this is a situation players should try to avoid, but I think it is a flaw in the game system. The issue could surely be fixed by implementing a mechanic that allows players to spend metal on something useful late in the game.

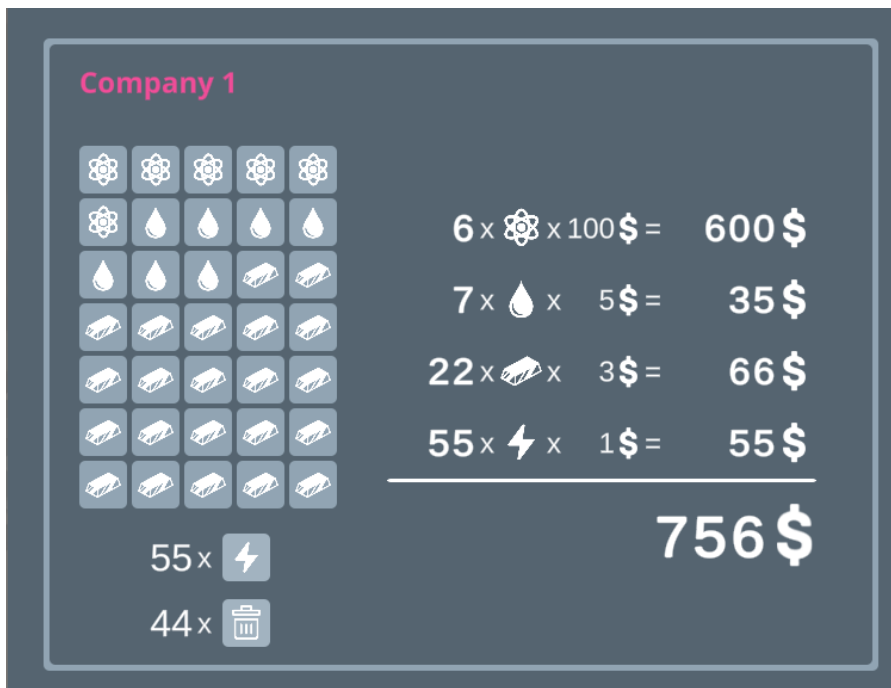


FIGURE 75
Metal overflow at
the end of the game

Passive Gathering

Passive gathering was definitely the mechanism that had the most problems to integrate nicely into the whole game system. Passive gatherers struggled to find a clear role next to the standard strategy of gathering fuel from the map to produce energy and ultimately cores.

On one hand, passive gatherers can be used as a supplementary option. Some players would build a Solar Panel to edge out a few more energy resources they needed in the short-term to afford their first core production. Similarly, late in the game a passive gatherer might produce a few more resources to give a marginal score advantage. However, most of the time, there was always some action that was more important to pursue for the standard gameplay strategy. There were area controllers to construct and dismantle, resource deposits to conquer and gather and production buildings to construct, protect, enhance and restart. Somehow, there was just never the right time to fit in the construction of a passive gatherer.

On the other hand, there was the possibility of going all-in on passive gathering. Players could totally ignore the standard strategy and choose to gather resources passively instead of from the map. In the beginning I feared that this strategy would heavily disincentivize area conflict, but that wasn't the case. Firstly, there was no passive gatherer for metal resources and thus conflict over metal deposits would always take place. Secondly, players going for a passive gathering strategy would try to create large, efficient bases of enhanced passive gatherers which were so powerful that opponents were basically forced to take aggressive action against them. Overall, this advanced strategy provided a nice alternative that enhanced the strategic possibility space of the game, but it was also a very all-or-nothing approach that didn't easily blend over into other strategies.

Ultimately, the game doesn't really need passive gathering. There is a good argument to be made that the mechanic goes against the grain of the core gameplay and should be cut out. However, the game also appears to be somewhat boring and one-dimensional without it. I think the better approach would be to find a way to integrate passive gathering better into the game, which should be a main focus for any further game design development.

A Couple of Issues

Overall, the third game design iteration was a great success. I had a strong vision of how a particular set of gameplay mechanics would come together to create a coherent whole and, for the most part, that worked out great. However, some of the emergent dynamics I hoped to see never came to fruition. For example, I anticipated a more interesting competition between players for the same resource deposits. As resource deposits now held multiple resources and could be gathered by both players at the same time, I envisioned situations in which players would fight over the supply by adding additional gatherers or utilizing Enhancers or even Disruptors. In the end, it turned out that the

amount of resources in any given deposit or cluster of deposits never justified the costly investment of such actions. It just wasn't worth it to enhance or disrupt a gatherer that would be obsolete after a few turns.

Another somewhat disappointing fact was that players would only need a very small amount of production buildings. As mentioned before, games like Anno and Settlers were a big inspiration of mine. In those games players build complex production chains with dozens of buildings. In the current version of the game, however, players needed only three production buildings at the maximum – two Power Plants and one Fusion Factory.

While the gameplay was a lot more dynamic and unpredictable than before, it still remained a bit rigid. This was to be expected from an open information game that uses no randomness at all. Like chess, it is very cerebral and serious in nature. The game lacks the excitement of surprises and the potential pressure alleviation of chance mechanics. This rigidity is further enhanced by the pre-designed maps that provide a very curated playspace and can't possibly generate any unexpected circumstances. Of course, this type of play is a question of taste and some players are really into strategy games with complete open information. For my taste, however, this game could use a few interesting chance mechanics or other forms of mixing up the game to force players to react and adapt their gameplay.

Sadly, my initial goal of creating a somewhat finished product was ultimately not realized. While the current prototype is fully playable by two human players on one computer, it is definitely still in a developmental state. The game lacks any type of structuring menus, options and explanations and loading up a map is only possible via the map editor. Nevertheless, I'm very satisfied with the final state of the game for the time being and if I had more time for further development I'd probably continue to improve the game design before worrying about finalizing a public build.

2.7.10 Workflow Evaluation

The workflow of the third design iteration was the same as in the prior cycle. I did an extensive conceptualization phase focusing on the most glaring issues of the preceding version of the game and continued with prototyping, playtesting and evaluation phases with increasing frequencies.

In retrospect, I think the scope of this design pass was just right. It was a big leap forward, but it wouldn't have made much sense to continue with more little steps, as I needed to change multiple interdependent systems at once to achieve my next vision. The volume of new additions to the system was large but still manageable. I always felt in control of the design space because I had built such a strong foundation of understanding in the first two iterations. Ultimately, the big step forward was a great success and resulted in a considerable improvement of the game.

I'm particularly proud of how I was able to simplify and straighten out some of my conceptual ideas that didn't work out right from the beginning. I managed to let go of the complicated maintenance and storage ideas and found a great solution for the automatic resource distribution issue in the restarting mechanic. Furthermore, I skillfully streamlined the turn counter mechanism and its corresponding mechanics to form a coherent gameplay subsystem. Generally, I didn't settle for any suboptimal solutions and put in the work to iron out any significant issues. The result was a game system that generated a well-structured, compelling strategy gameplay experience.

For me, the most enjoyable experience of this design iteration was the balancing phase. The game was now a complex interdependent web of distinct systems and mechanisms and their respective parameters could be freely configured to adjust the overall gameplay experience. At that stage, it felt like the difficult work was already done and the rest of the design process was just about playing with values to slowly narrow down the final arrangement.

One downside of such a large step forward was of course the enormous workload. I spent a lot of time on game design, technical implementations, visual communication, playtesting and balancing. In the end, I still feel like Version 3 of the game isn't truly finished and I didn't even start with updating the AI opponent. It's always somewhat disappointing to end a project in development limbo, but at least, with the help of a highly effective workflow, I was able to develop a great playable prototype and a remarkably compelling strategy game in just a few months' time.

2.8 Version 3: AI Concept

As discussed before, I had decided to use the last weeks of development to work on the game design of Version 3 and consequently didn't have any time left to update the game AI. The game system and decision making process had gotten so complex, that the implementation of a competent AI player would have probably taken months. However, because my initial goal was to produce a final prototype that included a functional AI opponent, I felt a little disappointed. Therefore, I decided to at least create a basic concept of the next AI version and its most essential components.

2.8.1 Core Utility System

The utility system at the core of the game AI would basically stay the same and each possible action would still require an individual utility function. As the game now contains 13 distinct buildings, which can be placed, dismantled and in some cases restarted, the amount of basic actions would rise from 8 to over 30. Therefore, it's reasonable to assume that the technical implementation of utility functions, as well as influence maps, would require a more robust, modular architecture, possibly involving some form of node-based editor. Furthermore, in order for the AI designer to effectively work on the configuration of this amount of actions, the AI system would need more advanced debugging, as it wouldn't be feasible anymore to work with logs in the console and parameters in the inspector. Thus, the system would require a sophisticated, custom-made editor window that could visualize all important information in a compact and comprehensible fashion.

In order to modulate the additional decision factors of the more advanced gameplay of Version 3, many new considerations would have to be implemented. It is hard to anticipate in advance which exact factors would be necessary to create convincing decision making and I can only make guesses. Human players often work towards specific goals, like starting a production building. Therefore, it would perhaps make sense that when a Power Plant needs fuel to be restarted, the utility function of building a Fuel Pump would increase. This abstract concept of needs could perhaps enable the AI player to handle a wider variety of unique situations.

As the game now included more complex action sequences and branching tactical decision paths, I am fairly certain that the simple staircase design of the first AI version wouldn't be practical anymore. For the staircase to work, it must be always reasonable to take an action if it is possible. Now, however, players often deliberately choose to not execute a powerful move because it would interfere with their current goal. For example, players abstain from building an Enhancer because they are saving up energy to restart a Fusion Factory. While this sounds very much like the AI system would require some sort of explicit goal, I am sure this concept could also be factored into the utility functions in a more abstract fashion.

2.8.2 Additional AI Systems

I'm confident that I could develop a functional AI player with well-configured utility functions alone, but in order to create a game AI that can handle some of the more advanced tactics and strategies, as well as provide more interesting variety and playstyles, additional AI features would surely be required to complement the core system.

Meta-System

I'm absolutely sure that the game AI would benefit greatly from a higher-order system that is concerned with long-term strategies and playstyles because the core utility AI system only decides about one action at a time, but players are now making larger-scale choices, such as playing overly aggressive or going for the risky passive energy strategy.

The meta-system would pick a strategy based on simple, preset map parameters or even a sophisticated map analysis and then influence specific utility functions. For example, an aggressive strategy would add score bonuses to offensive moves like building the Disruptor. The passive energy strategy would give a boost to the utility of building Solar Panels and also significantly decrease the scores of building Fuel Pumps and Power Plants.

Such a system could also be used for additional purposes, like modulating difficulty levels or giving companies distinct personalities. Therefore, I think that a modular, data-driven approach to the technical implementation would be most suited to afford optimal design control and flexibility.

Specialized Sub-Systems

As the game gets more complex and players are tasked to think about more unique questions, the game AI might need to utilize specialized sub-systems, because forcing every aspect of the decision making process into the core utility functions could unnecessarily complicate the system and make it harder to work with. This was actually already happening in the first version of the AI, as one of the most critical decision factors – spatial reasoning – was already covered by the separated influence map system.

One gameplay dynamic that would surely need its own decision making system is the puzzle-like combinatorics of building shapes. It is now really important to carefully consider in which sequence and arrangement buildings can be placed within the limited building space. Players can't just place their buildings anywhere, they need to think ahead. The problem, however, is that the utility AI isn't able to think ahead. A possible solution would be a separated sub-system that works out optimal building placements by creating a graph of all possible future combinations, scoring the branches and giving the best first node to the utility function that handles the placement action. While such an approach wouldn't directly modulate the definite plans that human players often form, it could make sure that every placement results in a remaining building space that is well-suited for further development.

2.8.3 Optimizing for Player Experience

It would certainly be quite difficult to develop a competent game AI for the third version of the game, yet achieving this task might only be half the battle. The prior game version already proved that an AI player that plays the game in an optimal way might not be what's most fun for the human player to interact with. Being dominated by a clinical, merciless opponent might even turn out to be incredibly frustrating. Therefore, I'm convinced that dedicated modulation mechanisms would be required to optimize the game AI for the player experience

In order to be fun, a game needs to be challenging, yet winnable. Of course, players will always have different levels of competence, so the game would need difficulty levels, which means the game AI would require a sophisticated mechanism to make suboptimal decisions. Usually, this is done by picking not the best, but one of the best scored actions, presumably with a weighted random selection. For the AI of my game, however, this would not only apply to the core utility system, but also to the various sub-systems, which provide preselected data to it. For example, an influence map would also have to provide not the best, but one of the best tiles for something like a building placement.

Maybe this method would be sufficient enough to create interesting situations for players to shine and get the upper hand against the AI player. Another more advanced approach would be to design the game AI to make deliberate mistakes that can be exploited by human players. After all, there is nothing more satisfying in this game than to pull off a successful aggressive move like stealing a valuable building.

Of course, the AI system could be optimized in a variety of ways. In the end, however, the development context and available resources would have to decide how far the AI for Version 3 should be pushed. Possible further development plans are discussed in the next section.

2.9 Outlook

While I am very glad to conclude the bachelor project with a well-rounded final prototype, I definitely feel like Version 3 is just an intermediate milestone of a longer journey. Next to the obvious task of updating the AI opponent, there is still so much I want to add to the game. Because I see such great potential for the project, I have gathered a number of ideas of how the development could potentially continue.

2.9.1 Possible Milestones

The first apparent milestone would be to create a polished, final Version 3 of the game. This would, of course, include a fully functional, well-configured AI opponent. Then, I would really like to organize an open tournament to let enthusiastic players battle it out against each other and the AI opponent. This would hopefully give valuable feedback about the ultimate appeal and replayability of the game, as well as interesting insights into more advanced gameplay strategies. Such an event could either be a great conclusion to the project or the starting point of more serious development.

The next milestone would be to develop an alpha release version. While I could imagine extending the core gameplay with another resource and a couple of buildings, the main difference from the prior version would be a curated single player experience. This would include some form of simple metagame that leads the player through a series of play sessions on different maps, starting with easy tutorial levels and progressing in difficulty to end with a challenging climax. This version would be published online on popular indie dev websites to reach a wider audience and gather honest feedback from real players with no affiliation to myself. Players would earn a final score and achieve a ranking on a public leaderboard. This would hopefully incentivize some people to play the game multiple times, which would help to judge the general replayability of the game. Ultimately, a public release would not only be a great experience in and of itself, but also serve as a great portfolio piece for my professional career.

If the public engagement and feedback for the alpha release version would be overly positive, I could even think of a full commercial release. I'm fairly sure, however, that in order to be successful, the game would need significant enhancements in almost all areas. This would include the core and metagame, but most importantly the user experience and the visuals. Such an undertaking would probably take over a year, even with the help of a few additional developers.

2.9.2 Gameplay Development

Extended Version 3

As of now, there is only one mechanism that I would like to add to the third version of the game system and it is a solution for the metal overflow problem that happens late in the game when players dismantle a lot of buildings. In order to provide a viable option to spend metal resources, I would introduce a special building – the Export Shuttle –, which can be constructed for a small amount of metal and fuel resources and will immediately send a number of resources, picked by the player from their inventory, off the planet. These resources would, of course, be scored at the end of the game. With the right balance of parameters, the Export Shuttle should not only fix the metal overflow problem, but also increase the general strategic depth of the late game by providing another interesting option that has to be coordinated with the dismantling process. Additionally, a constructible spaceship would give some nice variety to the set of buildings and underline the space travel theme.

Version 4

I already have a quite extensive backlog of ideas for a potential next iteration of the game's design. Many concepts naturally involve building functionalities. For example, the current set of building actions – placement, dismantling and restarting – could be expanded by upgrading, which would modulate the parameters of building effects and processes. Upgrading would be an interesting option that further complicates the decision making process and it would probably be easy and fun to come up with helpful bonuses, like increased area ranges, faster gathering speeds and lowered restarting costs. Another idea would be buildings with global effects, which would basically represent technologies. Such buildings would be very valuable and consequently create natural milestones that further structure the gameplay phases and provide intermediate goals to strive for.

One of my initial ideas, which sadly never got included into the game up until now, was the auction trade mechanic. Because I still think it would complement the current gameplay very well – providing another interesting way of interaction between players and breaking up the somewhat monotonous gameplay rhythm –, it would be my top candidate for an additional gameplay system for the fourth version of the game. Every 10 turns or so, a trade ship would visit the planet and enable players to bid for resource purchases and sell-offs. By making the set of available resources random, the mechanism would also introduce a bit of unpredictability to the game, which would force players to adjust their game plans and hopefully result in a more compelling strategic experience.

Of course, there are still many other gameplay ideas I didn't yet include in the game, such as dedicated gameplay structure mechanisms, procedural map generation and additional offensive actions. In the end, however, I should stick to the principles of

adaptive game design and not put too many new elements into a single design cycle. The exact scope of a potential fourth or even more additional iterations would have to be determined by my specific milestones.

2.9.3 Meta Game

Ultimately, the core gameplay shouldn't be extended too much, as it was always supposed to be experienced in a relatively short play session. However, in order to create a more elaborate strategy game that can be played for many hours, I would like to develop a sophisticated meta game in which the player controls one of multiple companies that are sent to a distant solar system to exploit the resources of its orbiting planets. The game would be structured as a sequence of core games that are played on the different planets, each having individual characteristics, like specific map layouts or resource distributions, which offer different gameplay experiences. For example, one planet could have very little fuel resources, forcing players to utilize passive gatherer, while another might have no atmosphere, rendering the Gas Filter building useless. Potentially the players would be offered multiple possible levels to pick from each round to provide interesting decisions and accommodate for individual preferences and playstyles. The other, computer-controlled companies would also have unique characteristics and competence levels and serve as distinct rivals to compete against. The money made from each core game would translate into the meta game and accumulate as a global highscore that judges the player's performance and incentivizes replay.

3. Discussion

3.1 Workflow

Initially, I had set out to develop a quite challenging project involving complex game systems and complicated game AI. Additionally, I made the claim that game design and AI development could be parallelized. In order to achieve such ambitious goals, I had planned to utilize an iterative workflow and adapt to challenges as I go. This section is a discussion of how successful this approach ultimately turned out to be.

3.1.1 Adaptive Game Design Process

Throughout this project, I applied the iterative design workflow described in section 2.1.3 to develop my game in a highly adaptive manner. Overall, this workflow approach turned out to be as effective as expected. I went through a number of distinct design phases, which helped to structure the whole development process (Figure 76) and I always felt in control of the design space because I only had to focus on one step at a time. I never determined any specifications earlier than necessary, which afforded me



FIGURE 76
Distinct phases of
the development
process

great flexibility to adjust to any unforeseen challenges and problems were detected and solved early and efficiently. Ideas got filtered, adjusted or discarded and new concepts were generated and integrated to evolve the game in the right direction. Ultimately, the efficacy of this adaptive game design process is proven by the increasing quality of each progressive version of the game.

Distinct Game Versions

One of the most helpful elements of my particular adaptive workflow was the definition of distinct game versions. The fact that a version was just one step in a progressive sequence and not the final game meant that I had a much easier time to limit the scope of my concepts and avoid any feature creep. Each version had its own particular goal or theme and I could put concentrated focus on singular gameplay aspects, which resulted in a much higher design quality. The goal of finishing the current version always provided a great milestone to aim at and work towards, which increased my motivation and drive noticeably. I never felt discouraged or overwhelmed and, in contrast to many of my previous game projects, there was no moment in the development process in which I wanted to quit or start over again.

The most crucial advantage of defining distinct game versions can actually be found in the transition between the versions. At the end of developing one version, the final nature of its last prototype incentivizes a deep analysis and reflection that normally wouldn't happen with any intermediate prototype. Of course, the findings of these thorough examinations are then incredibly valuable in feeding the next cycle. Because the conclusion of one game version is experienced as definitive, starting the next one actually feels like a brand new beginning. These fresh starts helped my mind to be more open about letting go of old ideas and thinking about new paths to follow. In my previous game projects, I often struggled to find the right moment for a decisive change of direction, even if the evidence for its necessity kept accumulating, but with the clear cuts between distinct game versions, that moment occurred naturally and I was able to achieve a degree of adaptability that I had never experienced before.

One example of a drastic shift in design direction happened between the second and third version of the game. I already had a fairly concrete plan of which changes I wanted to implement with the next iteration, mainly concerning a more advanced economy, but when multiple playtesters highlighted their strong desire for more aggressive gameplay actions, I scrapped the concept and thought up more offensive mechanics like stealing and disrupting. Unsurprisingly, these exact changes were very well perceived once the new version could be playtested.

Cyclic Development

In practice, my cyclic development process didn't perfectly fit the one-dimensional model of iterative game design. While each distinct game version generally went through the four stages of the cycle – conceptualization, prototyping, playtesting

and evaluation – , the real process resembled more of a fractal pattern (Figure 77). Inside each loop were many smaller loops. The prototyping stage included many little playtests, which were quickly evaluated to adjust the design and once I started to transition into the playtesting stage, the findings were still analysed to immediately change the prototype. Over the course of one large cycle, the nested inner cycles tended to get faster because the design changes transitioned from substantial revisions to more subtle configurations.

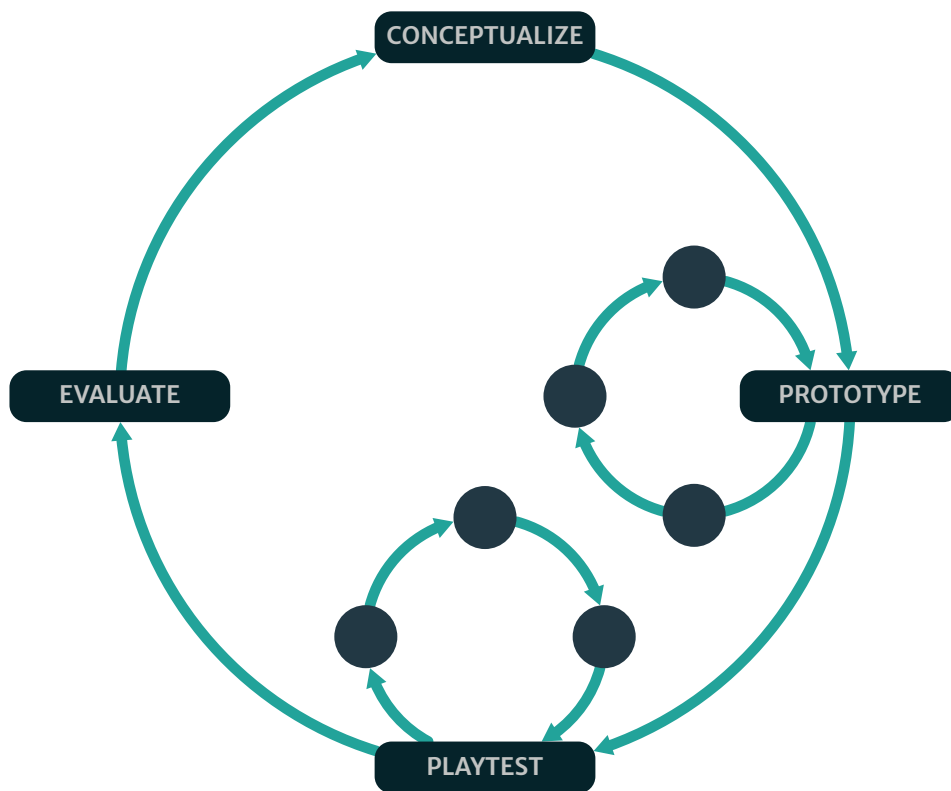


FIGURE 77
Nested loops inside
the iterative game
design cycle

Regardless of whether an iteration cycle was global or nested, it's scope had to be determined in its conceptualization stage. I have found that this decision can be a tricky balance. Smaller steps are easier to handle because the lower amount of new design substance is easier to oversee and control, but small steps also tend to be slower and sometimes certain transformations are only possible by combining multiple changes to the game, which requires a bigger leap forward. However, a large volume of alterations can potentially overwhelm the designer, lead to suboptimal decision making or simply take too long to process, which basically negates the core benefits of the itera-

tive workflow – regular playtests, feedback and conceptual readjustments. In the end, the right scope of each individual design cycle depends on the state of the game and the vision of the designer, but a general guideline would be to make it as big as necessary, yet as small as possible.

Evolutionary Design

When utilizing an iterative workflow, the game's design goes through an evolutionary process – it starts out small and simple, then organically develops to become larger and more complex.

The early phases provide a hotbed for experiments and idea radiation in which fundamental design issues are identified and rejected, while good aspects are selected for further development. At this stage, it is very easy to determine the quality of specific gameplay elements because there is no distracting complexity yet. One example in my development was the area control mechanic. The earliest version clearly showed that simplistic, absolute control would result in a strong first mover advantage and disincentivize spacial conflict. From this point on, I knew that I had to come up with a solution eventually. If I had made a more complex first prototype, including many more moving parts and focusing on a more advanced economy, I could have potentially overlooked this dynamic.

The simplicity of early prototypes also means that substantial design changes are still feasible and easy to implement technically. Therefore, it is crucial to make significant alterations as soon as possible. Once early developmental stages are finished, they form the basis for consecutive iterations, which generally keep the core systems, mechanics and properties intact and further adjustments become more about refinement. For example, Version 3 of my game retained the core loop of Version 2, yet expanded it with a more complex production chain.

Of course, the evolutionary process also has some disadvantages. Sometimes characteristics that once served a purpose can become problematic over time and are hard to get rid of. Early on in this project, I had spent a lot of time developing functionality for procedural map generation and I had gotten so used to the idea that I struggled to shift to custom-made map design. This is where the rigorous fresh starts of the distinct versions can help tremendously to allow for more drastic changes or complete redesigns of certain features.

Modular Idea Collection

One method that worked extremely well in conjunction with the adaptive design workflow was the creation of a modular idea collection at the beginning of the project. This catalogue included various potential gameplay systems and mechanisms, each with multiple possible implementation specifications, as well as a number of critical design questions and possible answers. For example, I had many different ideas on how area control could work – it could be static or dynamic, instantaneous or expanding and

there were different possible solutions for what would happen when two areas collide or when a building's control state changes. I documented all of these ideas and thoughts, but never decided on anything definitely.

Then, throughout the game design process, I could draw and pick ideas from this collection as needed. Some concepts were introduced at the beginning of new distinct versions, while others resurfaced at critical development phases to provide useful solutions to acute problems. At these moments, I could use my increased knowledge about the game to make more qualified decisions about detailed specifications. For example, at the time I added full and semi control to the third version of the game – in hopes of enhancing spacial conflict –, the findings that playtesters desired more aggressive gameplay actions led me to introduce the rule that full control trumps semi control, which enabled great offensive gameplay dynamics, such as cutting of control and stealing buildings.

Overall, having a rich collection of malleable ideas afforded me great flexibility in the design process and it turned out that many questions are actually fairly easy to answer when the time is right. In hindsight, any design approach that advocates starting with a definitive concept now seems foolish to me.

Playtesting

Possibly the most critical benefit of the iterative game design workflow is its emphasis on playtesting. Playtests are absolutely critical in game design, as only player feedback can give an authentic assessment of gameplay experience. The focus on distinct iterations made sure that I was always aiming for the next playable prototype – a goal that also incentivized me to reduce the scope of my designs. It took just a few hours to create my first paper prototype and playtesting it provided important findings that greatly influenced the rest of the game's development. This pattern continued throughout the whole game design process. In any given iteration, it was always the playtesting phase that generated the most precise knowledge about what works and what doesn't. It was this feedback that enabled me to make confident design decisions and evolve the game in the right direction.

3.1.2 Adaptive Prototyping

Of course, the adaptive nature of the iterative design workflow also applies to the game's prototype and each version is always an extension of its predecessor. In practice, this can have some disadvantages, as all the different gameplay systems have to be developed in conjunction. In the beginning, when the prototype was still very simple, I could focus on a few isolated mechanisms, such as area control and resource gathering, but as complexity grew, I had to focus on many more elements and their interdependencies and each addition to the game needed to be balanced against everything that was already established.

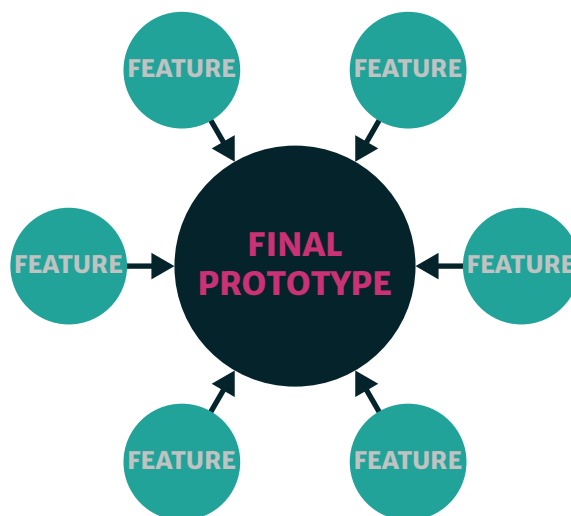
When multiple gameplay systems must be adjusted at the same time, it can be difficult to serve every need adequately. While systems with obvious problems, like the turn counter mechanics, easily forced my attention, other more subtle issues were harder to take care of. For example, the early area control rules weren't terrible, but merely mediocre and I had to consciously push myself to actively improve them. Other additions, like the resource types and the production chain of Version 3, were acceptable enough that I just kept them in their initial state without ever experimenting with alternatives and there is a good chance that further investigation would have led to interesting improvements.

Some of the ideas I considered to be integral parts of the initial vision, like auction trading, actually never made it into the prototype. The adaptive workflow led to such a significant expansion of complexity that I never got around to including them. Maybe an auction mechanism would have been great for the game and maybe it would have been even more interesting than the area control mechanic. Perhaps auctions and area control would have worked well together and players could have fought over control by bidding for it. Sadly, I never had the chance to find out. Of course, I could still introduce auction trading with the next design iteration, but it is possible that a late addition to the game would produce a vastly different result from what would have happened if I had prototyped it earlier.

Feature Prototyping

Interestingly, there would have been another approach to trying out different gameplay ideas – feature prototyping, a methodology in which designers create not one, but multiple small prototypes for a number of different gameplay systems. One interesting example is the game Spore. More than a dozen unique prototypes, which all focus on different design directions, were developed and can be found on the official website of the game ([Spore Prototypes], 2009).

FIGURE 78
Many feature
prototypes precede
the final prototype



It's interesting to think about what would have happened if I had created more specialized prototypes for my fundamental gameplay ideas. A specific prototype about area control, without any economic aspects, might have generated more dynamic control rules and possibly even convinced me to develop a game purely focusing on spacial conflict. On the other hand, a prototype about economic mechanisms, without distracting spacial dimensions, may have allowed me to discover more interesting production chains. Potentially, I could have developed simple prototypes for many more ideas, like building combinatorics or auction trading, but that would have probably taken too much time.

Ultimately, it is difficult to predict how my game would have developed if I had utilized feature prototyping. I do think, however, that if I had to start over, I would certainly do a few more initial prototypes instead of just one, especially because the early prototypes are generally quick to produce. Of course, in the end, all game systems have to come together to form a coherent whole, but I'm sure it would have been beneficial to explore the design space from a few more directions in order to make a more educated decision about the starting point for the final prototype (Figure 78).

3.1.3 Adaptive Programming

Another part of the development process that had to align with the adaptive workflow was the technical implementation and programming of the game. Generally, adaptive programming has the same strengths and weaknesses as any other adaptive process. The progressive evolution of the codebase means that every new addition is built on what was previously established, which is often beneficial as single tasks are generally smaller, easier to control and can rely on existing functionality. On the flipside, older parts of the code can become problematic when they don't fit new requirements, which can be especially frustrating when that code was intentionally designed to be as modular and extensible as possible. One example from my project was a complicated system for building effects, which was built to handle all possible effect types in a modular, polymorphic fashion and allowed for multiple effects per building. In the end, I actually never designed a building with multiple effects, but the functionality caused some annoying problems with the area control mechanics because I unnecessarily implemented full and semi control as separate effects.

Sometimes, when following the adaptive principle, I would develop new functionalities by utilizing existing program architecture or code components, even though I knew that wasn't the best solution. This practice could become quite awkward when it was done repeatedly for multiple consecutive development steps. One example was the area control algorithm discussed in section 2.7.2, which grew more and more complex and convoluted over time and ultimately became an absolute mess. The obvious solution to resist this process of code degeneration is to periodically refactor the program and the transition between game design iterations thankfully provides a perfect moment for it. I did this once after the completion of the second version of the game. I not only

managed to clean up and streamline the codebase significantly, but I also prepared it for the requirements of the upcoming AI development.

Now, after all the implemented changes for the third version of the game, the program would definitely benefit from another rigorous refactoring, but, of course, restructuring the codebase would require a lot of time and effort. Therefore, I think that the argument that an adaptive process leads to a lot of additional, potentially unnecessary work might be especially relevant when it comes to programming. This is probably the reason why this approach is more common among smaller projects that are generally easier to technically implement, while big projects, like AAA titles, usually work with more definitive game concepts, which carry less risk for requiring major revisions.

3.1.4 Integrated AI Development

My initial goal was to fully integrate the AI development into the game design process. On one hand, this parallel approach was supposed to make sure that the game AI was always up-to-date in order to guarantee perfect playability, which, as discussed before, is an indispensable part of the iterative design process. On the other hand, I was curious to see if such a tight coupling of AI and game design would result in any interesting cross-pollination of ideas. In retrospect, I must admit that I have failed to achieve this goal. The game AI wasn't always up-to-date and the final prototype doesn't even have a functional AI player.

Perhaps the most critical moment that led to the diminished role of the game AI was when I created the turned-based paper prototype. When I made this decision somewhat spontaneously, my full attention was on game design and I didn't realize the remarkable ramifications it would have on my objective of parallel AI development. I made the paper prototype turn-based because I wanted it to be playable by two human players in order to enable a rapid iterative design workflow. While this decision was immensely beneficial for the game design process, as I could quickly playtest and evaluate the most fundamental concepts of the game, it unintentionally negated the initial problem statement that the game AI would be necessary for an effective design process. If the game could be playtested with human players alone, there was no urgent need for an AI opponent. At first, I wanted the game to change to real-time, which would have potentially made human versus human play impossible (disregarding multiplayer networking), but such a huge gameplay shift never seemed convenient, as the game design progressed nicely as it was.

So interestingly, an attempt to increase the effectiveness of the game design process sidelined the AI opponent. As a result, I didn't manage to merge AI and game design as I had envisioned and I couldn't generate significant findings about the method. Nevertheless, I have one important takeaway – parallel AI development should be approached with caution because it will inevitably slow down the iterative design workflow

Alternating AI Development

Because there was no need for an up-to-date game AI, I could always push its development back and focus exclusively on the game's design until it was in a satisfying state. This led to an AI development rhythm that was alternating instead of parallel (Figure 79). First, I would develop a new design iteration of the game, utilizing human versus human play for testing and evaluation purposes and then, once the game design was finished, I would create the AI opponent to mirror the human players. Throughout this process, playing against the AI was only used to develop and configure the AI player itself and not to influence the actual design of the game.



FIGURE 79
Alternating
between game
design and
AI development

Generally, I think this approach to AI development can be very effective. It isn't as fine-grained as a fully integrated style, but it is still adaptive and benefits from the same advantages like the other discussed processes, as each version builds on what was already established and is therefore smaller in scope and easier to control. If I would start to develop the game AI for Version 3 of the game right now, I would probably greatly benefit from the knowledge I gathered while creating the previous AI version and could reuse and evolve many of the technical implementations such as the influence map system.

Furthermore, I have to admit that integrating AI development into the game design process of Version 3 seems very impractical in retrospect. I had so many things to manage simultaneously – the new gameplay mechanics, the technical implementation, the visual communication – that any concerns about the game AI would have likely overwhelmed me. Maybe I was naive to think I could develop the AI on the side, especially as I was the only developer on the project and had to do everything myself. Game AI is extremely difficult and regular context switching seems ill-advised. Maybe parallel AI development would be more viable for a team that has a dedicated AI designer, but for a small project alternating AI development appears to be more practical.

Of course, in practice, I only ever created one AI version. Arguably this happened because the project concluded with a game design pass and the second AI version would be next in line if the development continued. However, this circumstance exactly highlights one of my initial concerns with a development rhythm that always puts the game AI behind the game design – if the AI development is always the last thing to do, there is a good chance that it will have to be cut short in the end, which will result in a disappointing final outcome.

Influences on Game Design

As discussed in the introduction, I was very interested to see whether the AI development would have a substantial impact on the game's design. For example, reading about influence maps, before I actually started the project, convinced me that building placement was a great core mechanic for a game with an AI player. However, throughout the actual development process, there was surprisingly no other gameplay idea that spawned from working on the game AI. This might have resulted from the fact that the AI development and game design were more separated than initially envisioned, but I do think that I might have overestimated this potential in general.

One positive impact of including game AI in the project was that it incentivized me to keep the game simple. While I was designing the first two versions of the game, I definitely thought about how the AI would be able to handle potential gameplay mechanics, which effectively constrained the scope and complexity of my concepts. This effect was certainly welcome, as I generally tend to struggle with keeping things simple and manageable. Interestingly, when I started to design the third version of the game, I knew that I wouldn't be able to develop the AI player for it anymore, at least in the scope of this bachelor project, which consequently diminished my concerns about how the AI would handle new gameplay elements, resulting in a much more complex game design pass.

Another way in which the AI development influenced the game design was that it forced me to analyze and understand the game and how it is played in a very deep way. I have never done such a thorough decision making analysis for any game as before developing the game AI for this project. As it turns out, reverse engineering a human player results in a lot of valuable knowledge about a game system, which is obviously of great help in the design process.

Conclusion

One thing is for certain – developing game AI is very complex and difficult. I had to do extensive research and learn a lot about the topic to implement the AI opponent for the second version of the game, yet I still feel like a total beginner who doesn't know what he's doing. In hindsight, parallel development might have been naively ambitious and I'm still not sure if it would have been possible had I designed a game more suitable for this approach. However, it has become clear to me that every game will have unique requirements for its game AI, resulting in varying degrees of applicability for different development structures. Therefore, as long as the game design is the most important priority, it doesn't make sense to pick a specific AI development style beforehand. As my game's design process ultimately didn't require an up-to-date game AI, parallel development wasn't needed and an alternating rhythm was just fine.

3.2 Fields of Work

One of my main goals for this project was to work in a number of specific fields that interest me – mainly game system design, technical design (AI development) and game programming – to see which I prefer the most and potentially orientate the direction of my professional career accordingly. This section is a reflection on how I experienced each of those fields and what that might mean for my future as a game developer.

3.2.1 Game System Design

Overall, I'm very happy with my game design performance. I started out with a specific yet flexible vision, utilized a highly effective workflow and made remarkable progress, resulting in a satisfying final game prototype. My adaptive workflow afforded me great control over my design space and allowed me to make excellent decisions in a precise and goal-oriented manner. I was able to constantly improve the game by enhancing its most essential gameplay elements and ironing out its most glaring issues, while always making sure that all the individual components fit together to form a coherent whole.

This project has clearly confirmed to me that I enjoy designing games that are system-heavy and utilize abstract, distinct mechanics. I just love to design complex, interconnected systems and processes that players can interact with. What might be even more exciting to me than creating the fundamental mechanics of such games is the work that comes after – the addition, configuration and balancing of data-driven content. I also feel more motivated, confident and at home when working on system-heavy game genres, like strategy, management and simulation games, because I personally like to play them. All in all, it has become abundantly clear that I prefer system design over other types of game design, such as movement mechanics or player-centric aspects, like interfaces, controls or game feel.

I'm also really proud of my final game, even if I don't consider it complete yet. I was able to stay true to my initial vision and create an engaging strategy game that has gotten overwhelmingly positive feedback from playtesters. This success shows that I have grown significantly as a game designer throughout my studies and I finally felt like I knew what I was doing and everything worked out as it was supposed to. Nevertheless, there is still so much to learn, which is incredibly exciting. Overall, this project has definitely reassured me that actual game design is the field of work that I'm most passionate about.

3.2.2 AI Development

On one hand, I find game AI incredibly fascinating and really enjoy reading and learning about its theory, different methodologies, techniques and implementation styles. On the other hand, AI development can be extremely challenging in practice, as it involves complicated logic that is hard to understand, customize and implement technically. Additionally, the actual configuration of AI behavior is also very difficult, even with advanced debugging tools. So while I was generally very excited and motivated to develop the game AI for this project and enjoyed it most of the time, this ubiquitous level of difficulty made parts of the process quite frustrating and unpleasant.

Perhaps the most challenging part of AI development is the design and implementation of an adequate architecture, especially because it's quite difficult to find concrete resources to learn from, as most information about game AI online or in print media is either oversimplified or heavily generalized. Even after reading multiple books and analyzing a number of open-source architectures, I still had to put a lot of effort into creating my own custom implementation of a Utility AI system that could handle the relatively simple gameplay requirements of the second version of my game. However, the AI development became more enjoyable once I started to configure the actual behavior and there was something magical about seeing the AI player come to life and grow more competent with every adjustment. Again, as in the field of game design, it seems like I have a preference for the configuration of a system over its initial architectural design.

Finally, even though I stand by my decision to spend the last weeks of the project entirely on the game design of Version 3, I do have to admit that I'm a little disappointed that I didn't get to develop a second AI pass. Creating the AI player for the final version of the game would have been certainly intriguing, yet undoubtedly challenging, maybe even overwhelming, but ultimately a great experience. For now, however, I don't feel like I have spent enough time on AI development to give a final verdict on whether I would want to do it professionally. I could probably still see myself as an AI designer, but not as a specialized AI programmer, as this role seems to demand too much specialization in order to learn the required field-specific knowledge and technical skills.

3.2.3 Programming

I generally really like programming and enjoy the challenge of technically implementing the game systems and mechanics I develop. A lot of the time, game design and programming actually go hand in hand and influence each other, which can be an interesting and beneficial process. Often certain game design concepts remain rather vaguely defined until they have to be put into executable code. In my project, this happened with my idea about maintenance costs, which turned out to be rather impractical once I started to really think about its exact technical implications. Looking back at all the projects I did throughout my studies, I have definitely gotten better at anticipating technical concerns, which improved my ability to evaluate gameplay ideas significantly.

Therefore, I'm quite convinced that regularly programming game logic actually makes me a better game designer.

Sometimes, however, I just enjoy programming for its own sake. Once in a while, there is something relaxing about focusing on writing a program that has clearly defined specifications instead of making ambiguous decisions about gameplay mechanics with vague goals like creating fun for the player. Then again, programming can also be extremely challenging and frustrating, especially when the game grows more complex. I still feel like I'm lacking the educated knowledge to confidently handle complicated software architecture and advanced programming topics, like design patterns, abstraction, modularity and others. Because of this, I definitely want to develop my programming skills further and hopefully I will get the chance to learn from professionals in the future.

In the end, even though I'm really interested in programming and want to improve my skills, I'm quite certain that I'm more passionate about game design and ultimately wouldn't be happy as a pure programmer. However, I would definitely miss programming if I wouldn't do it at all and want to find a way to integrate it in my professional role as a game designer.

3.2.4 Working Alone

Developing a game project alone is definitely quite interesting. On one hand, having full control is obviously great and there is no need for time consuming coordination with other developers. I also really like to work on a game as a whole and not only on small, isolated components. As I generally get bored quite quickly when I have to do the same task for too long, being able to switch between different responsibilities throughout this project was great and, as discussed before, I especially enjoyed working on the game design and programming. On the other hand, there are also tasks that I don't like as much, like developing the visual communication, and I would have been happy to delegate this responsibility to an UX designer or artist. Of course, context switching between different tasks can also be quite challenging, especially when dealing with complicated topics like AI development and, at times, it's difficult to really focus on one thing. Therefore, I think the right balance for me would be to work on two or three fields that I'm truly passionate about, but not more.

While having full creative control over a game project is great, working all alone can also be quite tough. Having no team members means there is nobody to bounce ideas off, to help with problems and to keep up the spirit in troubling times. Over the course of my studies, having done so many group projects, I have definitely come to appreciate team members, as everybody has their own unique viewpoint and skill set that is valuable to the development process. Consequently, I don't think I would want to develop a full game all alone for an extended period of time.

3.2.5 Conclusion

This project has definitely helped me to further clarify where my enthusiasm is strongest within the wide spectrum of game development. The exact center of my interests lies somewhere in the game system design area, close to the technical domain (Figure 80). Therefore, if I had to be a specialized worker, I would want to be a game system designer. However, it is quite evident that I enjoy working on a broader range of tasks and would be happier having a job in which I can wear many hats.

FIGURE 80
My personal
interest curve in the
game development
spectrum

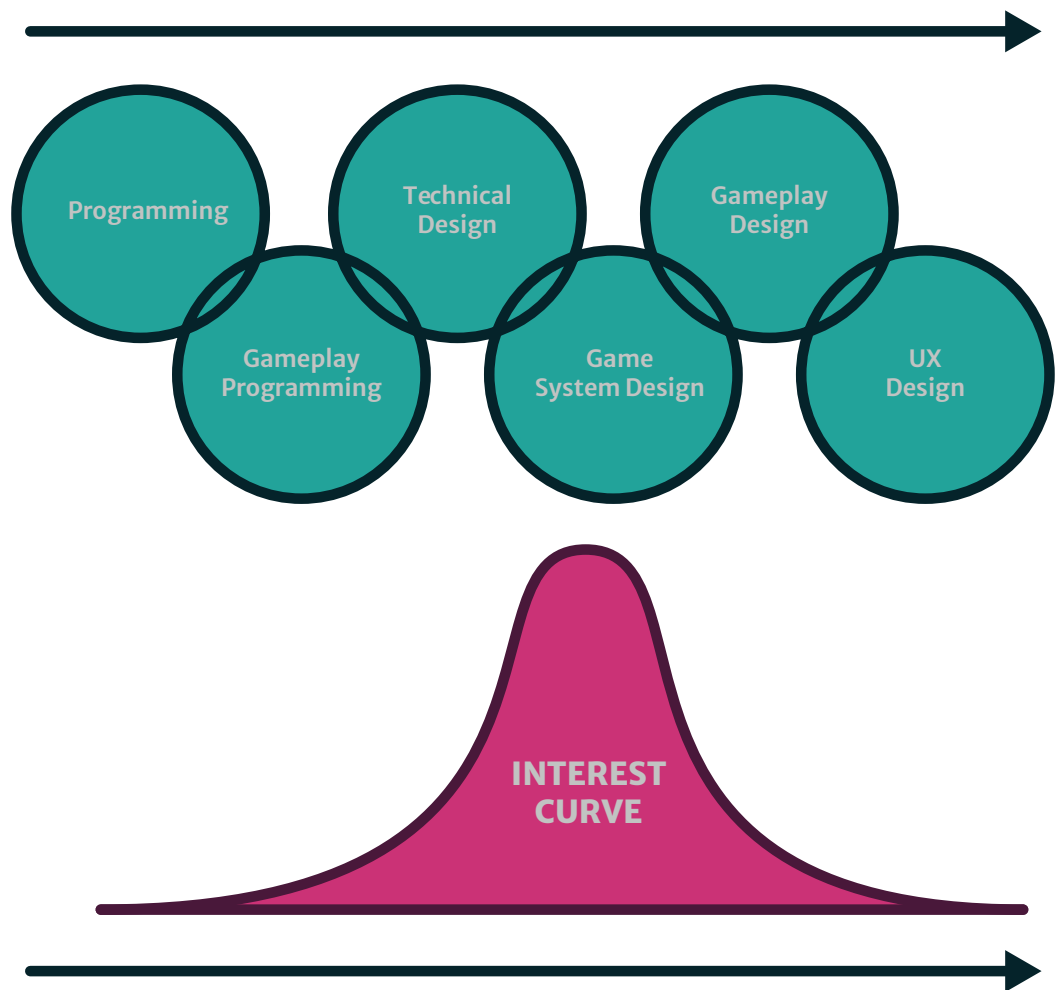


Figure References

- Figure 1** Iterative design process
Macklin, C., & Sharp, J. (2016). *Games, Design and Play: A Detailed Approach to Iterative Game Design*. Addison-Wesley.
- Figure 2** Building placement in Islanders
Islanders [Video Game]. (2019). Grizzly Games
- Figure 3** Castle exerting implicit map control in Age of Empires 2
Age of Empires 2: Definitive Edition [Video Game]. (2019). Xbox Game Studios
- Figure 4** Distinct provinces in Europa Universalis 4
Europa Universalis IV [Video Game]. (2013). Paradox Interactive
- Figure 5** Explicit territory control in Settlers 3
Settlers 3: History Edition [Video Game]. (2018). Ubisoft Entertainment SA
- Figure 6** Optimization of building space in Anno 1404
Anno 1404 [Video Game]. (2009). Ubisoft Entertainment SA
- Figure 7** Starting placement influencing the progression of the game in Catan
Mit Glück und Strategie zum „Siedler“-Weltmeister. (2014). Welt.
https://img.welt.de/img/newsticker/dpa_nt/infoline_nt/boulevard_nt/mobile133163652/9091623907-ci23x11-w1136/urn-newsml-dpa-com-20090101-141011-99-01107-large-4-3-jpg.jpg
- Figure 8** Players control tiles of a shared playspace in Terraforming Mars
Die Zähmung des roten Planeten. (2017). Neue Zürcher Zeitung.
<https://img.nzz.ch/2017/6/1/6523780f-a445-4bc4-9c30-3225d0a9c019.jpeg?width=1360&height=604&fit=crop&quality=75&auto=webp>
- Figure 11** 3-dimensional modulation of hexagonal tile grid
Patel, B. (2013). *Hexagonal Grids*. Red Blob Games.
<https://www.redblobgames.com/grids/hexagons/>
- Figure 15** The central generator blocks the view of players in Frostpunk
frostpunkgame.com. bit studios S.A.
https://www.frostpunkgame.com/wp-content/uploads/2019/09/frostpunk_screenshot_01.jpg
- Figure 17** Offworld Trading Company
offworldgame.com. Mohawk Games.
https://stardock.cachefly.net/www_offworldtrading_com-assets/store/otc_pleasure_dome_city.jpg
- Figure 19** Production chains of Anno 1800
Production chains. Anno 1800 Wiki.
https://static.wikia.nocookie.net/anno1800/images/c/ce/Production_chains_updated_jan_2021jpeg.jpeg/revision/latest?cb=20210122124613
- Figure 43** Influence map with floating point values
Mark, D., (2015). *Modular Tactical Influence Maps*. Game AI pro 2: Collected wisdom of game AI professionals. 343–364. CRC Press, Taylor & Francis Group.

References

- Adams, E., & Dormans, J. (2012). *Game Mechanics: Advanced Game Design*. New Riders.
- Brewer, D. & Graham, R. (2018). *Knowledge is Power: An Overview of Knowledge Representation in Game AI*. [Video]. Youtube.
<https://www.youtube.com/watch?v=Z6oZnDIgio4>
- Fan, K. (1953). *Minimax Theorems*. Proceedings of the National Academy of Sciences, 39(1), 42–47. <https://doi.org/10.1073/pnas.39.1.42>
- Gibbons, R. (1992). *A Primer in Game Theory*. Prentice Hall, Upper Saddle River, NJ.
- Graham, D. (2014). *An Introduction to Utility Theory*. Game AI Pro. Collected Wisdom of Game AI Professionals. 113–126. CRC Press/Taylor & Francis Group.
- Larman, C., & Basili, V. R. (2003). *Iterative and Incremental Developments. A Brief History*.
- Lewis, M. & Mark, D. (2015). *Building a Better Centaur: AI at Massive Scale*. [Video]
<https://www.gdcvault.com/play/1021848/Building-a-Better-Centaur-AI>
- Macklin, C., & Sharp, J. (2016). *Games, Design and Play: A Detailed Approach to Iterative Game Design*. Addison-Wesley.
- Maeda, J. (2006). *The Laws of Simplicity*. MIT Press.
- Mark, D., (2012). *AI Architectures: A Culinary Guide*. <http://intrinsicalgorithm.com/IAonAI/2012/11/ai-architectures-a-culinary-guide-gdmag-article/>
- Mark, D. (2009). *Behavioral Mathematics for Game AI*. Charles River Media, Course Technology, Cengage Learning.
- Nystrom, R. (2014). *Game Programming Patterns*. Genever Benning.
- Patel, B. (2013). *Hexagonal Grids*. Red Blob Games. <https://www.redblobgames.com/grids/hexagons/>
- Schell, J. (2008). *The Art of Game Design: A Book of Lenses*. Elsevier/Morgan Kaufmann.
- [Spore Prototypes]. (2009). Spore.com. <http://www.spore.com/comm/prototypes>
- Thompson, T. (2018), *The AI of Empire: Total War (Part 2 of 5) | AI and Games*. [Video]. Youtube. https://www.youtube.com/watch?v=KL_AAGSivbI

List of Tools

Throughout the development of my workpiece, I used the following tools and plugins.

SOFTWARE

- Unity
- Visual Studio
- Adobe Photoshop
- Miro
- Google Docs
- Google Sheets

Unity Plugins

- Odin – Inspector and Serializer (Sirenix)
- Master Audio: AAA Sound (Dark Tonic Inc.)
- Interface and Item Sounds (Cafofo)
- Modern UI Pack (Michsky)
- Ultimate Sound FX Bundle (Sidearm Studios)
- Easy Performant Outline 2D | 3D (Ruslan Kudriachenko)
- Advanced Dissolve (Davit Naskidashvili)
- Thread Ninja – Multithread Coroutine (Ciela Spike)
- DOTween (Demigiant)

Note Of Thanks

Thanks, to begin with, to my parents Kerstin and Andreas Mros for their abundant support, which enabled me to have a second chance to acquire my bachelor's degree.

Deepest thanks go to Prof. Susanne Brandhorst and Prof. Thomas Bremer for giving me the opportunity to study in the course of Game Design and creating such a special environment for interconnected learning and growth.

Furthermore, thanks to Prof. Jan Berger and all other educators for their dedicated teaching of specialized knowledge.

The warmest thanks go to my student peers for their support and comradery, teaching me that there is something to learn from everybody and making these three and a half years a special and pleasant experience.

Finally, thanks to everybody who helped me in the creation of my bachelor project. Thanks to Prof. Thomas Bremer for his helpful coaching and thanks to the many enthusiastic critics and playtesters who spent their valuable time to give feedback on my work.

Statement Of Authorship

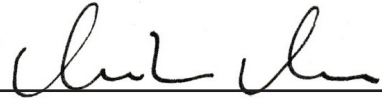
I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper.

I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content.

I am aware that the violation of this regulation will lead to failure of the thesis.

Berlin, 08.06.2021

Place, Date



Name



GAME DESIGN

DEHIVE

htw

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

