

UNIVERSITY OF PIRAEUS

DEPARTMENT OF INFORMATICS

**Fantasy Auto Battlefield,**

an Auto Battler Card Game

This dissertation is submitted for the degree of

*Bachelor in Software Engineering and Intelligent Systems*

Supervisor

Prof. Themis Panayiotopoulos

Candidate

Konstantinos-Zois Syrios

Piraeus, February 2022

Abstract

The subject of this dissertation is the development of the game *Fantasy Auto Battlefield*. *Fantasy Auto Battlefield* combines elements from the *Card Game* and *Auto Battler* genres, and more information about those will be given in the introductory chapter. During the development of the program, special emphasis was given to the creation of an easy-to-learn and easy-to-use method of card creation, so that possible future developers may have the necessary tools to easily create new cards.

In the second chapter, there will be an analysis of the reasons for using the Unity 3D game engine and an overview of some of its features that were used during the development of *Fantasy Auto Battlefield*.

In the third chapter, an overview of the game rules and goals will be given. *Fantasy Auto Battlefield* is a game that pits the human player versus an AI player, with both players playing cards until at least one of the bases is destroyed.

The subject of the fourth chapter, will be the design of the user interface and what needs and choices lead to its development. Included there, will be an overview of the general user interface, as well as the card design.

In the fifth chapter, there will be a detailed overview of the code base. A short description will be given for every script developed and then a more comprehensive analysis of the main scripts will follow. Moreover, in this chapter the AI behavior logic will be analyzed and an overview of the used assets will be given.

Following the above, a game session will be showcased, accompanied by pictures as the player makes progress during a match of *Fantasy Auto Battlefield*.

Keywords

Unity, Design Patterns, Card Game, Auto Battler, Board Game, Hexagons

Special Thanks

I would like to specially thank my friends for play testing my game and my bother for composing the music.

Contents

[Introduction 6](#_Toc98633978)

[The genre of the game 6](#_Toc98633979)

[Chapter 2 - Why use Unity? 7](#_Toc98633980)

[Chapter 3 – Rules and Gameplay 7](#_Toc98633981)

[Board Structure 7](#_Toc98633982)

[Terrain Types 8](#_Toc98633983)

[Playing Cards 8](#_Toc98633984)

[Units 8](#_Toc98633985)

[Buildings 9](#_Toc98633986)

[Spells 9](#_Toc98633987)

[Pawn Placement 10](#_Toc98633988)

[Frontline 10](#_Toc98633989)

[Mana 10](#_Toc98633990)

[Turn Structure 11](#_Toc98633991)

[Initiative 11](#_Toc98633992)

[Unit Movement 11](#_Toc98633993)

[Pawn Combat 11](#_Toc98633994)

[Victory Conditions 12](#_Toc98633995)

[Chapter 4 – User Interface 12](#_Toc98633996)

[Graphical Environment 12](#_Toc98633997)

[Heads Up Display 13](#_Toc98633998)

[Board UI 15](#_Toc98633999)

[Card UI 16](#_Toc98634000)

[Pawn UI 18](#_Toc98634001)

[Help Panel 19](#_Toc98634002)

[Chapter 5 – Implementation 20](#_Toc98634003)

[Overview of codebase 20](#_Toc98634004)

[Factories 20](#_Toc98634005)

[Info Containers 20](#_Toc98634006)

[Scriptable Objects 21](#_Toc98634007)

[Scripts 21](#_Toc98634008)

[Board Scripts 21](#_Toc98634009)

[Card Scripts 22](#_Toc98634010)

[Player Scripts 22](#_Toc98634011)

[UI Scripts 23](#_Toc98634012)

[HelpPanel 23](#_Toc98634013)

[MenuUI 23](#_Toc98634014)

[UI Observer-Subject 23](#_Toc98634015)

[Resources 24](#_Toc98634016)

[Analysis of main scripts – Board Generator 24](#_Toc98634017)

[Analysis of main scripts – Game Manager 26](#_Toc98634018)

[Analysis of main scripts – AIPlayer 28](#_Toc98634019)

[Analysis of main scripts – Factories 30](#_Toc98634020)

[Analysis of main scripts – FlyCamera 30](#_Toc98634021)

[Card Creation Method 31](#_Toc98634022)

[Presentation of the main graphical assets 32](#_Toc98634023)

[Chapter 6 – A playthrough of *Fantasy Auto Battlefield* 34](#_Toc98634024)

[Chapter 7 – Problems and future updates 35](#_Toc98634025)

[List of Used Assets (with links) 36](#_Toc98634026)

[Table of Images 37](#_Toc98634027)

[Bibliography 38](#_Toc98634028)

# Introduction

*Fantasy Auto Battlefield* was developed in the Unity 3D game engine using C#, and is a game that was built for the Windows operating system.

## The genre of the game

*Fantasy Auto Battlefield* combines elements from the *Card Game* genre and the more recent *Auto Battler* genre.

The first modern trading card game (TCG), *Magic: The Gathering* was published by *Wizards of the Coast*, Inc in 1993, and since then has inspired a great number of card games such as the more recent *Hearthstone* digital card game developed and publish by *Blizzard*.

Drawing inspiration from those predecessors, *Fantasy Auto Battlefield* supplies the player with a deck of cards, from which he draws cards and plays them by expending a resource called *Mana*. Each card may be a unit (or building) with its own stats, placed on the game’s board, or a spell card which affects the game’s board without leaving a “body” behind.

Some differences can be drawn from those games though, as the *Mana* resource here is generated by a flat amount each turn up to a maximum number that can be stored, whereas in the “*Magic: The Gathering”* for example the player needs to draw mana cards which after being played are replenished each turn. Another difference, is that in *Magic: The Gathering* and *Hearthstone* after the player draws all of his deck, they either lose or begin to take damage. In *Fantasy Auto Battlefield*, when the deck is empty, the player’s discard pile is shuffled into it.

*Fantasy Auto Battlefield* also draws inspiration from the *Auto Battler* genre. *Auto Battler* is a relatively new sub-genre of strategy video games that features chess-like elements where players place pawns on a board. The pawns after their placement, act autonomously according to some internal logic by moving and fighting opposing pawns.

The *Auto Battler* genre was popularized by games such as the *Dota Auto Chess* (which is actually a mod for the *Dota 2* game) and the *Teamfight Tactics* developed by *Riot Games.*

Similarly to those games, *Fantasy Auto Battlefield* has pawns that act autonomously although with more restraints as, for example, their movement is restricted on a single column of a hexagonal board.

# Chapter 2 - Why use Unity?

As mentioned before, the game engine that was chosen for the development of *Fantasy Auto Battlefield* is Unity 3D. Among its advantages, the Unity game engine boasts that it is royalty-free, a rich asset marketplace with even free assets, and, of course, the use of C# a modern and powerful Object-Oriented Programming (OOP) language.

Among the best features of Unity, I consider the *Monobehaviour* base class from which every Unity script derives. Using *monobehaviour’s* Awake(),Start(), and Update() methods, programming the behavior of game objects becomes effortless.

On the other hand, Unity also has two other features that greatly ease the usage of memory and those are the *Scriptable Objects* and the *Prefabs*. Both of those provide a way to implement templates for data fragments and game objects respectively, reminiscent of the *Flyweight* design pattern. Both *Scriptable Objects* and *Prefabs* were used extensively in the development of *Fantasy Auto Battlefield*.

Finally, a special mention should go to Unity’s extensive and comprehensive documentation which along with Unity’s forums, contain a wealth of information and problem-solving topics.

# Chapter 3 – Rules and Gameplay

In this chapter, the game design elements and rules of *Fantasy Auto Battlefield* will be explained.

A game of *Fantasy Auto Battlefield* takes place between a human player and an AI player. Both players take simultaneous turns that consist of five (5) phases. The goal of each player is to play cards and damage their opponent until their hitpoints reach zero (0).

## Board Structure

The board of *Fantasy Auto Battlefield* is made up of hexagonal tiles, each with its own unique terrain type.

For the dimensions of the board, after some experimentation, a five (5) width, eight (8) depth board was chosen. These dimensions provide a board of adequate size without slowing down the gameplay and offer interesting choices to the player.

The hexagons on the edges of the board in the depth axis, are of a special terrain type – the *base*. The *base* tiles represent the human player and their AI opponent.

When creating the rest of the board, each terrain type is allocated a percentage of tiles from the total board size, and then those tiles are placed randomly. This leads to a different map its game, which offers a strategic element to which the player needs to adapt differently in every game session.

## Terrain Types

*Fantasy Auto Battlefield*’s board is made up of the following terrain types:

* **Plains:** Plains have no effect.
* **Forest:** A pawn occupying a forest will have its defense increased.
* **Hills:** A pawn occupying a hill will have its attack increased.
* **Desert:** A pawn ending the turn on a desert tile will take damage.
* **Lake:**  A pawn ending the turn on a lake tile will heal a number of hitpoints.

## Playing Cards

The human player will have a deck filled with different cards of *Unit, Building* and *Spell* types. A card is drawn from the deck each turn and is played by paying its cost in *Mana*. When the player plays a card, it goes in the discard pile.

If the player’s deck is empty, and they try to draw a card then their discard pile is reshuffled into the deck.

The AI player does not use a deck, but instead chooses the cards that it will play from a tiered pool. More information on this process will be given in the fifth chapter, where a sub-chapter is completely dedicated to the AI’s behavior.

## Units

*Units* are one of the card types in *Fantasy Auto Battlefield*. When playing a *Unit* card, a new pawn is created at the chosen board tile representing the card. A *Unit* card has the following components/stats:

1. **Card Name**: The card’s name
2. **Mana Cost**: The card’s cost in mana
3. **Card Text**: The card’s special effect or flavor text
4. **Attack**: The card’s attack value
5. **Defense**: The card’s defense value
6. **Hitpoints**: The card’s hitpoints value
7. **Hex Pattern**: The card’s attack hex pattern (black: self, red: targeted hexagons)
8. **Card Type**: The type of the card

All *Units* also have the ability to move on the board. Each *Unit* moves one hex tile ahead in every move phase if that hex tile is empty.



Image - A Unit card

## Buildings

The second type of cards that the player will find in *Fantasy Auto Battlefield* are *Buildings*. *Buildings* have identical components and stats to the *Unit* cards, but most importantly they can not move. That makes the playing of *Building* pawns an important strategic decision, as placing them in the wrong board tiles may hamper the player.

## Spells

The third and final card type of *Fantasy Auto Battlefield* is the *Spell* type. *Spells* do not place a pawn on the board when played, and thus they do not have *Attack, Defense and Hitpoints* values. Instead, they have an immediate effect on the board in the form of drawing cards or destroying a pawn, etc.

## Pawn Placement

When the player decides to play a *Unit* or *Building* card, after paying its mana cost, they must decide where to place the resulting pawn. The player must always place their pawns on an empty board tile, adhering to the *Frontline* rule (see next sub-chapter). The players always have the choice of placing their pawns on the empty tiles of their base, as the opponent’s pawns never move onto them.

## Frontline

The *Frontline* rule dictates on which board tiles a player may place their pawns. The *Frontline* starts at the player’s base and expands as *Units* move on the board up to a maximum.



Image - Frontline depiction

For example, when the board is empty the player may place pawns only on base tiles (row 0). If the player owns pawns on the board that are on row one (1), he can also place pawns on that row. This effect maxes out at row two (2), as this was found to be a pretty balanced constraint for the mechanic. It also should be noted that pawns on greater rows (e.g., row 4), set the frontline to the maximum value.

## Mana

As mentioned before the main resource of the game and what the human player uses to play cards is the *Mana*. The player earns a number of *mana* at the beginning of their turn and can have stored up to a maximum of ten (10).

## Turn Structure

During a game of *Fantasy Auto Battlefield,* the players take turns that consist of 5 phases. Those are:

**Upkeep Phase:** The human player draws a card from their deck and gain mana, and also the initiative is changed (see next sub-chapter). Note: if the player’s deck is empty, the discard pile is reshuffled into it.

**Standard Phase:** The players decide which cards they will play. Each card decided upon must be paid for with mana for the human player. Then, the placement of the card’s pawn, or targeting of the card’s effect takes place, after which the card is moved to the discard pile.

**Move Phase:** The *units* on the board will now execute their auto-moves.

**Combat Phase:** The pawns now will attack any, and all possible targets.

**End Phase:** Effects that take place at the end of each turn are activated now (e.g., the desert tile terrain effect).

## Initiative

In case of ties during a game, such as *Units* of both players wanting to move in the same board tile, initiative is taken into account to determine priority. Only one player *has the initiative*, symbolized by the green coin, visible in *Image 2.* The *initiative* token changes hand every turn during the *Upkeep Phase.*

## Unit Movement

During the **Move Phase** all *units* will move one (1) board tile ahead in their column. The order in which the *units* will move is the following:

* The *units* of the player that has the *initiative* move first
* The *units* that have the greater distance from their base will move first
* No *units* may pass through friendly or enemy pawns
* No *units* may enter the enemy’s base tiles

## Pawn Combat

During the **Combat Phase**, all pawns attack simultaneously all available targets. Each pawn will compare its attack value with the defense of its targets. The attacker will deal as damage the differential between *Attacker’s attack* and *Defender’s defense* or a minimum of one (1), whichever is greater.

Every pawn that reaches zero (0) hitpoints be removed from the board.

The pawns have a *hex pattern* that they are able attack to. For example, a pawn may be able to attack the board tiles directly in front of it and behind it. Another pawn may be able to attack the three frontal board tiles.

When a pawn reaches the enemy base, it will deal 1 damage each turn to the base, even while attacking other pawns.

## Victory Conditions

A game of *Fantasy Auto Battlefield* ends when at least one of the players’ bases reaches zero (0) hitpoints.

# Chapter 4 – User Interface

The two main purposes that drove the decisions for the User Interface’s (UI) design, were simplicity and readability. In this chapter, the UI will be showcased using screenshots and explaining some of the logic behind its design.

## Graphical Environment

For the graphical environment of the game, a feeling of playing a board game was desired. To achieve this, a 3D room was created containing a table on which the game board is set up. Plenty of lighting in the room and warm colors on the wooden and fabric textures lead to a comfortable and enjoyable experience during the game.

The player can fly around the room using the camera controls (that will be explained in the next chapter) bound only by some constraints -free movements is only allowed within the bounds of a cube, smaller than the room and above the game board’s height- so that the camera movement does not interfere with the gameplay experience.



Image - Graphical Environment

## Heads Up Display

The Heads Up Display (HUD) of *Fantasy Auto Battlefield* is split into three (3) main parts.

Firstly, there is the upper part of the HUD. It contains the information about the hitpoints of the human player and how many mana points they have. It also states in which game phase the game is in.



Image - HUD upper part

The second part is where most of the information useful for the player is displayed. This part can also be split into three (3) parts of its own:

1. The Card Info section where information will be displayed when the player clicks on a pawn on the game board.



Image - HUD, Card Info section

1. The Tile info section where information will be displayed when the player clicks on a tile on the game board.

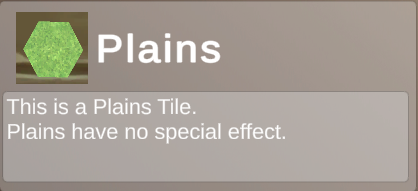


Image - HUD, Tile Info section

1. The menu section where the following buttons are displayed:
   1. **Toggle Hand Button:** it shows or hides the player’s hand of cards.
   2. **Play Card Button:** when the player has selected a card and wants to play it, they will click on this button.
   3. **End Phase Button:** this button is used to move to the next phase of the turn.
   4. **Open Menu Button:** this button opens the menu that the player can use to exit the game or view the *Help Panel.*

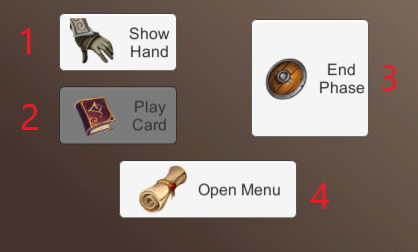
****

Image - HUD, Button section

And last but not least, there is the game log where the progress of the duel is recorded in text format. The game log can also be minimized using the upper right button as displayed below:

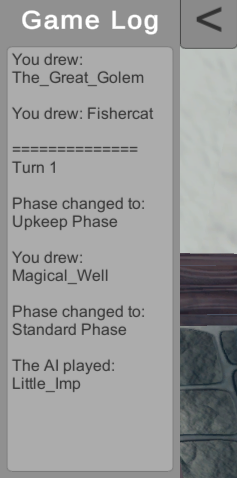


Image - HUD, Game Log

## Board UI

The *Board UI* is a rather small category that contains the display of the bases’ hitpoints, the human player’s deck counter and the *Initiative Token.*



Image - Base's Hitpoints



Image - Deck Counter



Image - Initiative Token

## Card UI

The components of a card’s UI have been previously shown in *Image 1*, so here instead there will be a comparison between a *Unit/Building* and a *Spell* card.



Image - Building Card

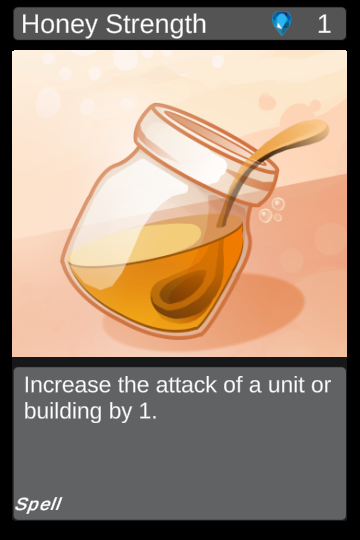


Image - Spell Card

As you can see the *Building* card has virtually no differences with *Unit* card. The difference between *units* and *buildings* is that the former can move.

On the other hand, the *spell* cards are “empty” in comparison. They only display the name, the cost and the effect of the card.

## Pawn UI

Each pawn is represented on the board with the game object depicted in the following image. The most prominent feature is the card’s artwork and the only thing that is displayed are the card’s current stats. This makes the state of the board extremely readable and easy to understand what card is each pawn.

The pawn’s stats are displayed with the same order as on the cards (i.e., Left: *Attack,* Upper Right: *Defense,* Down Right: *Hitpoints*).

*Note: The color of the pawn’s stats changes to green, white, and red when the current value is higher, equal, or lower than the original values respectively.*



Image - Pawn

## Help Panel

Finally, the player has access to a *Help Panel* in which information about the game can be read, accompanied by useful images.



Image - Help Panel

The player can browse through the entries of the *Help Panel* using the buttons at the bottom of the screen. The *Help Panel* was coded using Scriptable Objects for the entries, in order to be very flexible and extendable.

# Chapter 5 – Implementation

## Overview of codebase

I tried to design *Fantasy Auto Battlefield* to be as uncoupled as possible so new features and cards could be implemented easily. For the new cards specifically, I developed a workflow that enables someone to easily create a card (without a special effect) within minutes even without prior knowledge of the codebase. More information about this process will be given in one of the following sub-chapters.

I would also like to mention the usage of some design patterns such as Singletons, Factories, and Observer-Subject. In summary Singleton was used when global access of one instance was desired, such as in the Game Log, Factories were used for the creation of instances for the different cards during the game, and an implementation of the Observer-Subject pattern was used to send messages in a succinct way to the UI observers.

In the following table you can find all the code files of the project organized in alphabetical order and by folder, with short descriptions of their functionality:

|  |  |
| --- | --- |
| Factories | |
| BuildingFactory | This class was made following the Factory design pattern and its purpose is to create pawns of Building cards |
| SpellFactory | This class was made following the Factory design pattern and its purpose is to activate the effects of Spell cards |
| UnitFactory | This class was made following the Factory design pattern and its purpose is to create pawns of Unit cards |
| Info Containers | |
| CardType | Enumeration of the different card types for the game |
| GamePhases | Enumeration of the game phases |
| HexDimensions | Simple class that contains two necessary dimensions of the hexagon assets used as tiles for the game |
| HexPatternCodes | Enumeration of the different combinations of hexes that make up a targeting pattern for a card |
| TileHeight | Enumeration of the different heights that the hexes of the board will have (not used in the current implementation of the game) |
| TileType | Enumeration of the different terrain types available for the board's hex tiles |
| Scriptable Objects | |
| BuildingCardData | This Scriptable Object is used for storing data about the game's different building cards |
| SpellCardData | This Scriptable Object is used for storing data about the game's different spell cards |
| TileData | This Scriptable Object is used for storing data about the game's different tile types |
| UnitCardData | This Scriptable Object is used for storing data about the game's different unit cards |
| Scripts | |
| FlyCamera | This script implements the controls of the camera which the player uses to navigate the game’s 3D space |
| GameManager | This is the most central class of the game. It contains the logic of the game: it initializes the state of the session, and  progresses through the game phases |
| GameManager\_PartialHelpers | GameManager was implemented as a partial class so that the majority of methods could be in a separate file so that the logic flow was clearer in the main file |
| SFXController | Simple Monobehaviour to control when and which SFX is played |
| Board Scripts | |
| BoardGenerator | This class is responsible for generating and instantiating the game’s board |
| BoardManager | Provides access to the Board Generator and a static method to translate coordinates of array into coordinates of 3D board. |
| HexTile | This class is used to encapsulate the functionality of the board’s hexes. |
| Card Scripts | |
| Building | Building is a monobehaviour inheriting from the Spawnable class and is attached to the gameobject pawns that represent a Building card |
| Card | Card is a dummy class that both Spawnable and Spell classes inherit from |
| IEffectOnDeath | Interface for Cards that have an effect when they die |
| IEffectWhenSpawning | Interface for Cards that have an effect when they enter the game |
| IEffectWithTargetWhenSpawning | Interface for Cards that have an effect on the hex where they spawn |
| PawnStats | Small UI script that displays the stats (Health, Attack, Defense) of the pawn and their changes |
| Spawnable | Spawnable is monobehaviour, inheriting from the Card class and is attached to the gameobjects that represent a Unit or a Building card, as the respective Unit and Building scripts inherit from this class. |
| Spell | Spell is monobehaviour inheriting from the Card class and is attached to the gameobjects that represent a Spell card |
| Unit | Unit is monobehaviour inheriting from the Spawnable class and is attached to the gameobjects that represent a Unit card |
| Player Scripts | |
| AIPlayer | AIPlayer is a monobehaviour that inherits from the Player class.  It represents the opponent of the user. |
| Deck | The Deck class contains the cards used by the human player in the game. |
| DiscardPile | The DiscardPile class contains the cards owned by the human player that are discarded (e.g., overdrawn or played cards) |
| Hand | The Hand class represents the hand of the human player, i.e., contains their drawn cards. |
| HumanPlayer | HumanPlayer is a monobehaviour that inherits from the Player class.  It represents the user that plays the game. |
| Player | Player is a monobehaviour from which the HumanPlayer and AIPlayer inherit.  It contains some shared functionality such as the taking of damage. |
| UI Scripts | |
| BoardUI | This class is responsible for the UI of the Board.  It also is a UI observer in order to receive messages. |
| CardInHand | Each CardInHand contains the behavior for each of the five cards, that the human player may have drawn. |
| GameLog | The GameLog is used to print messages about the duel in the UI at the top right corner of the screen. |
| MainUI | This class is responsible for most of the UI including the card info and tile info areas.  It also is a UI observer in order to receive messages. |
| HelpPanel | |
| HelpPanel | HelpPanel contains the behavior of the game's help pages |
| HelpPanelEntry | A scriptable object that contains the information of the help panels |
| MenuUI | |
| CreditsPanel | UI class that loads the credits text and displays it in the appropriate panel |
| MenuBackground | UI class responsible for the carousel of images on the main menu |
| MenuButtons | UI class with the behavior of the main menu buttons |
| UI Observer-Subject | |
| IObserverUI | Interface for the UI classes that need to be observers of the Subject UI class |
| SubjectUI | SubjectUI is a class that its static methods can be called to notify all the UI observers |
| UIEvent | A UIEvent object contains the information necessary for observers to process a UI message when it is fired. |
| Resources | |
| CardCatalogue | This is a very important class that contains all the cards in the game in entries that include their name and type. |

For more details there are documentation comments inside the code files.

In addition to the above, in the resource folder there are the scripts of each individual card but adding them in the table would not offer more information to the reader. Most card scripts are empty (when the card has no effect) or is card specific only.

## Analysis of main scripts – Board Generator

The BoardGenerator script has the role of creating the board for each session of *Fantasy Auto Battlefield*. Most of the fields it has are serialized in the Unity Editor so the developer can easily change the parameters of the board’s generation. For example, the size of the board, the percentages of the tile types, and the Prefabs of the hexagonal tiles are all serializable.

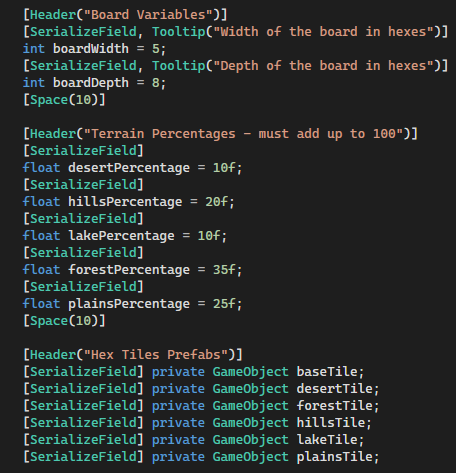


Image - Board Generation Parameters

In summary, the steps of generating a new board are:

1. Populating a pool with tiles dictated by the type percentages given;
2. Shuffling the pool;
3. Instantiating the board.

First, the pool is populated:

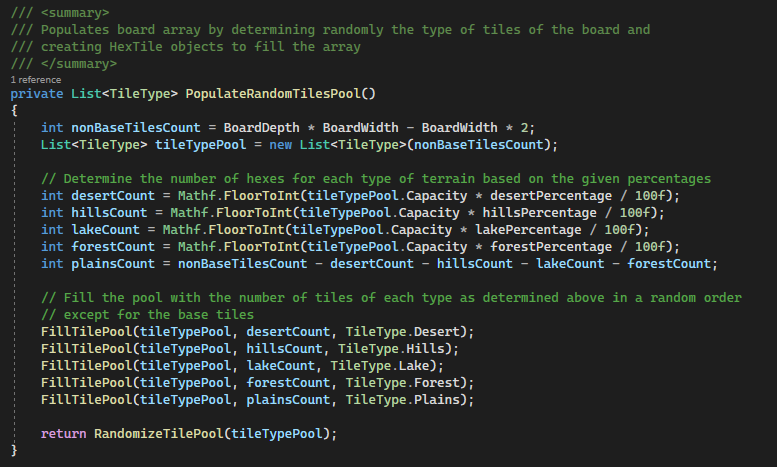


Image - Populating the Board Generator pool

Then it is shuffled, and finally game objects are instantiated in the 3D scene for each tile in the board array.

## Analysis of main scripts – Game Manager

As expected, the Game Manager class holds one of the most important roles in *Fantasy Auto Battlefield*. It is the central hub of game logic; it initializes the game state at the start of the session and oversees the smooth cycling of the game’s phases.

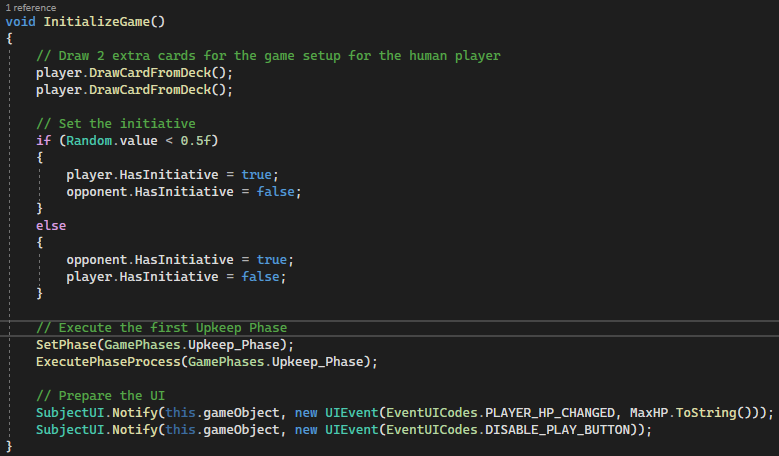


Image - Game state initialization by the Game Manager

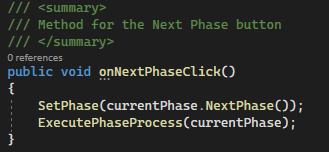


Image - Cycling through the Game Phases

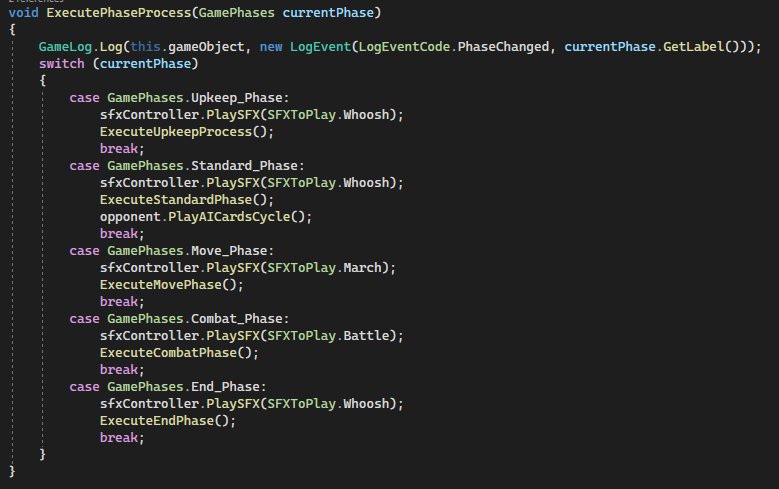


Image - Handling of each Game Phase by the Game Manager

Because of the sheer volume of code necessary for the Game Manager, the file quickly became difficult to handle and understand. For this reason, Game Manager was implemented as a partial class. This allowed to move most of the methods (e.g., the swapping of initiative, highlighting the tiles that contains pawns, moving of the *Units*, etc.) to another file, so the main Game Manager could stay clean, readable and easy-to-understand.

The partial file perhaps could be split further, into more files, but this was not the necessary for the current scope of the project.

## Analysis of main scripts – AIPlayer

In this important sub-chapter, the behavior of the Artificial Intelligence (AI) opponent will be explained. Although the AIPlayer script inherits from the Player class, same as the Human Player, it does not share the need for components such as hand, deck, discard pile, mana, etc. That is because the AI that was implemented decides what to play from an available pool of cards each turn without paying for them.

It decides on which cards to play semi-randomly. It should be noted that the AI adheres to the *Frontline* rule of the game and does not “cheat” in that regard. Also, the AI player always plays in the standard phase, before the human player giving the advantage of knowledge to the user.

The pool of cards from where the AI sources its cards is divided in three (3) tiers (low, mid, high). Depending on the number of turns played, the AI decides on a number of cards to play that may be from the same or different tiers. For example, on the fourth (4th) turn of the game, the AI may decide to play two (2) cards from the Low Tier or one (1) card from the Mid Tier. Those decisions, are taken randomly by using coin tosses. There is also twenty five percent (25%) chance for the AI to not play anything for a particular turn. This helps alleviate the constant pressure and the percentage was finalized through game testing to provide some challenge, but also not to feel overwhelming.

Finally, in regard to the choice of cards to play that depend on the turns’ count, two (2) implementations were developed, the *single* and the *cycle*.

In the *single* implementation, after reaching a particular turn number, the AI continuously decides between the choices of the late turns’ waves.

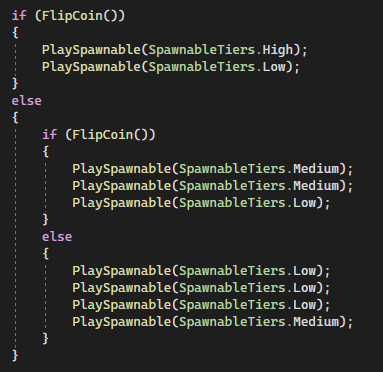


Image - Late turns’ waves in the Single implementation of the AI decision making process

In the c*ycle* implementation, after reaching a particular turn number, the turn counter is reset, so the AI returns to deciding between the choices of the early turns’ waves.

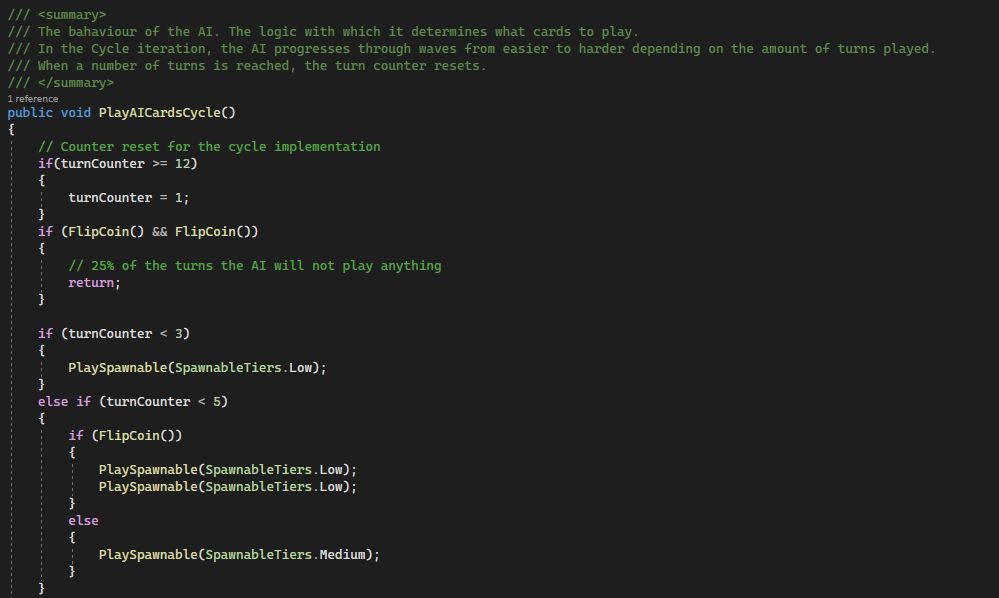


Image - Section of the Cycle Implementation of the AI decision making process

## Analysis of main scripts – Factories

The three types of cards (unit, building and spell) are categories that encompass many implemented cards. In order to provide an easy-to-use interface for the generation of the pawns and spell effects, the Factory design pattern was used. This enables the developer to create instances of the cards by only using the name of the card and the board coordinates they wish to affect (e.g., where to place the pawn).

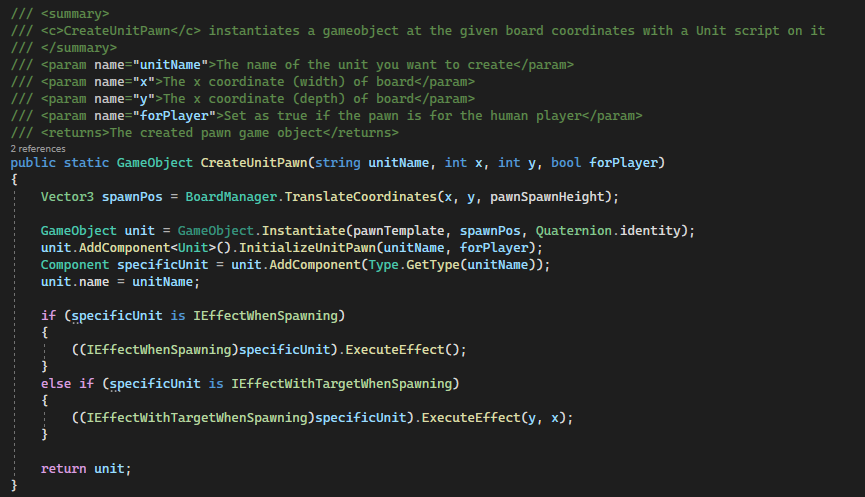


Image - Implementation of the Unit Factory

## Analysis of main scripts – FlyCamera

Finally, I would like to mention the FlyCamera script. The base of this script must be credited to *Windexglow (*[*https://gist.github.com/McFunkypants/5a9dad582461cb8d9de3*](https://gist.github.com/McFunkypants/5a9dad582461cb8d9de3)*)* and I made some additions to better suit the needs of *Fantasy Auto Battlefield*.

The base was that of a free fly camera script, but it was needed to constraint the player within some coordinates so that the camera movement did not impede the gameplay.

The area that the player will be constrained is an invisible cube above the table where the game’s board is set up.

## Card Creation Method

The card creation method was of great focus for this dissertation. The goal was that the process would be easy with minimal knowledge of the codebase and that it would be fast. The process can be distilled in the next five (5) steps:

1. Create a new **folder** in the Resources/Cards/{Units||Buildings||Spells}/ path with the name of the new card
2. Inside the **folder** right-click and create a new ScriptableObject of the suitable type (e.g., UnitCardData) with the name of the new card
3. Inside the **folder** create a C# script with the name of the new card
   1. If you want the card to have an effect, the script can implement the IEffectWhenSpawning, IEffectWithTargetWhenSpawning, or IEffectOnDeath interfaces
4. If you are creating a unit or a building, you will need to create a material with the name of the new card in the **folder**. Add as albedo texture the same sprite used in the ScritableObject of step two (2).
5. Finally, add the card to the catalogue by opening the CardCatalogue file and adding the line:

*Notes:*

1. *IEffectWithTarget currently can only target the hex chosen for the deployment of the card;*
2. *if you are creating a spell remember to call "Destroy(this.gameObject);" at the end of the effect; and*
3. *these instructions can be found in a text file in the Resources/Cards folder.*

## Presentation of the main graphical assets

Here, the main components of the graphical assets will be presented. All of them were found in the Unity Asset Store. A more detailed list with links will be included in one of the appendices.

Firstly, and most important is the collection of card artwork by REXARD. This is the only non-free asset, but a similar art style across the cards was very important, and the collection was humorous and with great variety so it was perfect for *Fantasy Auto Battlefield*. A single license was purchased for the asset package through the Unit Asset Store.

Secondly, the table and tablecloth compose the main background of the game, thanks to the somewhat aerial view of the camera. Both assets are included in the “Dinner Table” asset package by Evgenia.



Image - The Dinner Table asset package by Evgenia

Finally, the second most important aspect of the *Fantasy Auto Battlefield,* after the cards themselves, are the hexagonal tiles of the game. For those, after sculpting a base in the free and open-source program Blender (<https://www.blender.org/>) and adding to it a texture from the Unity Asset Store, there was the need for the model and texture of the top. Thankfully, the asset package “HexLands – Low Poly Style” by RRFreelance/PiXelBurner was exactly what was needed.

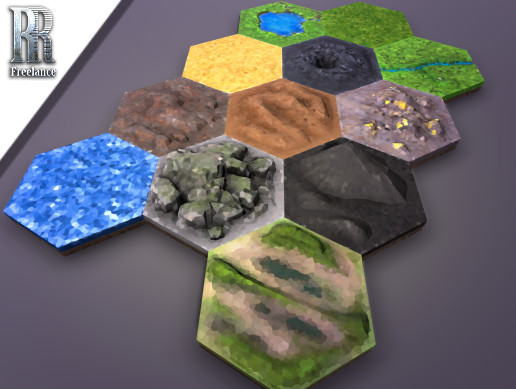


Image - Hexlands - Low Poly Style asset package by PRFreelance/PiXelBurner

# Chapter 6 – A playthrough of *Fantasy Auto Battlefield*

# Chapter 7 – Problems and future updates

The main problems that *Fantasy Auto Battlefield* currently face are the bugs that were not detected before the final stages of the development of this project (although some of them were found and were documented using a Bug Report template), and the coupling of the targeting logic with the board tiles.

I will focus mainly on the latter problem. During the development of the targeting system (i.e., on which tile should a pawn be placed, what tile should a spell affect), a convoluted process was formed that depends on a collider component that each board tile has. This allows only for a single target/tile to be chosen per time. A more general system was not implemented to account for multiple targets and that lead mainly to the problem of spell cards only targeting single tiles (with the extra constraint that those tiles must be occupied by a pawn).

An ideal future update should implement a revised targeting system that allows for multiple targets and does not create constraints for cards.

Another imperative update for the ideals of this project is the implementation of special effects as keywords (e.g., drawing a card as **summon 1).** This would enable the creation of a streamlined workflow for adding special effects to cards making the *Card Creation Process* even easier with minimal understanding of the codebase. Currently, if someone wants to implement a special card effect, they must understand the part of the code that is relevant to the effect they desire (in the previous example of drawing a card, they would need to find the method that implemented the draw a card functionality).

# List of Used Assets (with links)

**Special Mentions:**

1. [Music by Panagiotis Syrios](https://www.youtube.com/channel/UCDEUqssp1V-4YPjRfcDmjgg)
2. [TCG Cards Pack by REXARD](https://assetstore.unity.com/packages/2d/textures-materials/tcg-cards-pack-63019)
3. [Base of Camera Movement Script by Windexglow](https://gist.github.com/gunderson/d7f096bd07874f31671306318019d996)

**Unity Asset Store:**

1. [Card shirts Lite by Saji](https://assetstore.unity.com/packages/2d/gui/card-shirts-lite-165698)
2. [Dinner Table by Evgenia](https://assetstore.unity.com/packages/3d/environments/fantasy/dinner-table-55180)
3. [Fantasy Skybox FREE by Render Knight](https://assetstore.unity.com/packages/2d/textures-materials/sky/fantasy-skybox-free-18353)
4. [Free Simple GUI by Cryptogene](https://assetstore.unity.com/packages/2d/gui/free-simple-gui-185448)
5. [FREE Stylized PBR Textures Pack by Lumo-Art 3D](https://assetstore.unity.com/packages/2d/textures-materials/free-stylized-pbr-textures-pack-111778)
6. [Free Trees by Ada\_King](https://assetstore.unity.com/packages/3d/vegetation/trees/free-trees-103208)
7. [HexLands - Low Poly Style by RRFreelance / PiXelBurner](https://assetstore.unity.com/packages/2d/textures-materials/tiles/hexlands-low-poly-style-133586)
8. [Medieval Tavern Pack by 3DeLucas](https://assetstore.unity.com/packages/3d/props/furniture/medieval-tavern-pack-112546)
9. [Old Coin by Gnarly Potato](https://assetstore.unity.com/packages/3d/props/old-coin-49530)
10. [Procedural fire by Hovl Studio](https://assetstore.unity.com/packages/vfx/particles/fire-explosions/procedural-fire-141496)
11. [RPG inventory icons by REXARD](https://assetstore.unity.com/packages/2d/gui/icons/rpg-inventory-icons-56687)

**Sound Effects:**

1. [Foley\_Impact\_Metal\_Movement\_Mono.wav by Nox\_Sound](https://freesound.org/people/Nox_Sound/sounds/569973/)
2. [Whoosh.wav by crackles04](https://freesound.org/people/crackles04/sounds/369698/)
3. [distant horns by DANMITCH3LL](https://freesound.org/people/DANMITCH3LL/sounds/218488/)
4. [Victory and Defeat SFX by Panagiotis Syrios](https://www.youtube.com/channel/UCDEUqssp1V-4YPjRfcDmjgg)

# Table of Images

[Image 1- A Unit card 9](#_Toc98630919)

[Image 2- Frontline depiction 10](#_Toc98630920)

[Image 3 - Graphical Environment 12](#_Toc98630921)

[Image 4 - HUD upper part 13](#_Toc98630922)

[Image 5 - HUD, Card Info section 13](#_Toc98630923)

[Image 6 - HUD, Tile Info section 13](#_Toc98630924)

[Image 7 - HUD, Button section 14](#_Toc98630925)

[Image 8 - HUD, Game Log 14](#_Toc98630926)

[Image 9 - Base's Hitpoints 15](#_Toc98630927)

[Image 10 - Deck Counter 15](#_Toc98630928)

[Image 11 - Initiative Token 15](#_Toc98630929)

[Image 12 - Building Card 16](#_Toc98630930)

[Image 13 - Spell Card 17](#_Toc98630931)

[Image 14 - Pawn 18](#_Toc98630932)

[Image 15 - Help Panel 19](#_Toc98630933)

[Image 16 - Board Generation Parameters 24](#_Toc98630934)

[Image 17 - Populating the Board Generator pool 25](#_Toc98630935)

[Image 18 - Game state initialization by the Game Manager 26](#_Toc98630936)

[Image 19 - Cycling through the Game Phases 26](#_Toc98630937)

[Image 20 - Handling of each Game Phase by the Game Manager 27](#_Toc98630938)

[Image 21 - Late turns’ waves in the Single implementation of the AI decision making process 28](#_Toc98630939)

[Image 22 - Section of the Cycle Implementation of the AI decision making process 29](#_Toc98630940)

[Image 23 - Implementation of the Unit Factory 30](#_Toc98630941)

[Image 24 - The Dinner Table asset package by Evgenia 32](#_Toc98630942)

[Image 25 - Hexlands - Low Poly Style asset package by PRFreelance/PiXelBurner 33](#_Toc98630943)

# Bibliography

1. Nystrom, Robert. *Game programming patterns*. Genever Benning, 2014.
2. Gamma, Erich, et al. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.
3. Unity Documentation - <https://docs.unity3d.com/Manual/index.html>
4. C# Documentation - <https://docs.microsoft.com/en-us/dotnet/csharp/>
5. Hexagonal Grids from Red Blob Games - <https://www.redblobgames.com/grids/hexagons/>