A-Level Computer Science NEA

Basic Wars

Robert Choi

Table of Contents

[1. Analysis 4](#_Toc129948397)

[1.1. Intro to end user 4](#_Toc129948398)

[1.2. Game Background 4](#_Toc129948399)

[1.3. Game Rules 4](#_Toc129948400)

[1.3.1. Set up 4](#_Toc129948401)

[1.3.2. Turns 4](#_Toc129948402)

[1.4. Requirements of the end user 7](#_Toc129948403)

[1.4.1. Starting the game 7](#_Toc129948404)

[1.4.2. Map 7](#_Toc129948405)

[1.4.3. Players 7](#_Toc129948406)

[1.4.4. Artificial Intelligence (AI) 8](#_Toc129948407)

[1.4.5. Graphical User Interface (GUI) 8](#_Toc129948408)

[1.5. Research 8](#_Toc129948409)

[1.5.1. Framework 8](#_Toc129948410)

[1.5.2. Other Implementations 8](#_Toc129948411)

[1.5.3. Algorithms 9](#_Toc129948412)

[1.6. Objectives 11](#_Toc129948413)

[1.6.1. Menus 11](#_Toc129948414)

[1.6.2. New Game 11](#_Toc129948415)

[1.6.3. Game Map 11](#_Toc129948416)

[1.6.4. Save/Load game 11](#_Toc129948417)

[1.6.5. Players 12](#_Toc129948418)

[1.6.6. Artificial Intelligence (AI) 12](#_Toc129948419)

[1.6.7. Attacking 12](#_Toc129948420)

[1.6.8. Tiles 12](#_Toc129948421)

[1.6.9. Selecting/Manoeuvring units 13](#_Toc129948422)

[1.6.10. Units 13](#_Toc129948423)

[1.6.11. Game 13](#_Toc129948424)

[1.7. Prototyping and Modelling 14](#_Toc129948425)

[1.7.1. Simple Class Diagram 14](#_Toc129948426)

[1.7.2. Storing and retrieving game data 14](#_Toc129948427)

[2. Documented Design 16](#_Toc129948428)

[2.1. Overview 16](#_Toc129948429)

[2.2. Algorithms 16](#_Toc129948430)

[2.2.1. Common Variables 17](#_Toc129948431)

[2.2.2. Creating Unit Sprites 17](#_Toc129948432)

[2.2.3. Creating Tile Sprites 17](#_Toc129948433)

[2.2.4. Generating random points 18](#_Toc129948434)

[2.2.5. Getting Adjacent Tiles 20](#_Toc129948435)

[2.2.6. Map Generation 20](#_Toc129948436)

[2.2.7. Dijkstra Pathfinding 23](#_Toc129948437)

[2.2.8. Player Actions 24](#_Toc129948438)

[2.2.9. Saving/Loading Games 29](#_Toc129948439)

[2.2.10. Player Income 31](#_Toc129948440)

[2.2.11. Player Loop 31](#_Toc129948441)

[2.3. Classes 32](#_Toc129948442)

[2.3.1. Graphics 32](#_Toc129948443)

[2.3.2. Entities 33](#_Toc129948444)

[2.3.3. System 38](#_Toc129948445)

[2.4. Data Structures 40](#_Toc129948446)

[2.4.1. Graph 40](#_Toc129948447)

[2.4.2. Lists 40](#_Toc129948448)

[2.4.3. Dictionary 41](#_Toc129948449)

[2.4.4. Priority Queue 41](#_Toc129948450)

[2.5. User Interface 41](#_Toc129948451)

[2.5.1. Main Menu 41](#_Toc129948452)

[2.5.2. New Game Menu 42](#_Toc129948453)

[2.5.3. In Game 42](#_Toc129948454)

[2.5.4. Paused Menu 44](#_Toc129948455)

[2.6. Saving/Loading Games 44](#_Toc129948456)

[2.6.1. Saving Requirements 44](#_Toc129948457)

[2.6.2. Save File Examples 45](#_Toc129948458)

[2.6.3. Classes for saving and loading 46](#_Toc129948459)

[2.7. AI 47](#_Toc129948460)

[2.7.1. AI model 47](#_Toc129948461)

[2.7.2. States 47](#_Toc129948462)

[2.7.3. Logic Loop 50](#_Toc129948463)

[3. Technical Solution 51](#_Toc129948464)

[3.1. Areas of interest 51](#_Toc129948465)

[3.2. Code 51](#_Toc129948466)

[3.2.1. BasicWarsGame.cs 51](#_Toc129948467)

[3.2.2. Graphics 66](#_Toc129948468)

[3.2.3. Entities 68](#_Toc129948469)

[3.2.4. Enums 100](#_Toc129948470)

[3.2.5. System 101](#_Toc129948471)

[4. Testing 119](#_Toc129948472)

[4.1. Game Menus 119](#_Toc129948473)

[4.2. Unit Movement 120](#_Toc129948474)

[4.3. Unit Capture 120](#_Toc129948475)

[4.4. Unit Resupply 121](#_Toc129948476)

[4.5. Unit Attack 121](#_Toc129948477)

[5. Evaluation 122](#_Toc129948478)

[5.1. Evaluation of Objectives 122](#_Toc129948479)

[1.6.1. Menus 122](#_Toc129948480)

[1.6.2. New Game 123](#_Toc129948481)

[1.6.3. Game Map 123](#_Toc129948482)

[1.6.4. Save/Load game 123](#_Toc129948483)

[1.6.5. Players 124](#_Toc129948484)

[1.6.6. Artificial Intelligence (AI) 124](#_Toc129948485)

[1.6.7. Attacking 125](#_Toc129948486)

[1.6.8. Tiles 125](#_Toc129948487)

[1.6.9. Selecting/Manoeuvring units 126](#_Toc129948488)

[1.6.10. Units 126](#_Toc129948489)

[1.6.11. Game 127](#_Toc129948490)

[5.2. End-User Feedback 127](#_Toc129948491)

[5.3. How the program can be improved 128](#_Toc129948492)

[5.3.1. Artificial intelligence 128](#_Toc129948493)

[5.3.2. Saving/Loading 128](#_Toc129948494)

[5.3.3. Code Changes 129](#_Toc129948495)

[5.3.4. Online Multiplayer 129](#_Toc129948496)

[5.3.5. UI and Animations 129](#_Toc129948497)

# Analysis

## Intro to end user

Dylan Canavan is an avid gamer and enjoys various genres of games. He particularly enjoys turn based strategy games and is very familiar in their general mechanics.

However, most turn-based strategy games that Dylan comes across are rather complex and not very user-friendly in design. He believes that there should be a wider range of simpler turn-based games available to allow for people to get into the genre more easily.

Advance Wars is a game that Dylan played as a child on the Game Boy Advanced. Unfortunately, it is not available on modern machines and Dylan believes that it should be for more people to try.

To understand the requirements of the end user, I interviewed Dylan to find out what he wanted from the game (1.4). This information was then used to inform the research (1.5), objectives (1.6) and modelling stages of development (1.7).

## Game Background

Advance wars is a turn-based strategy game for 2-4 players. The objective of the game is to defeat the enemy armies. To win against an enemy army, the enemy HQ must be captured.

The game should have an easy-to-understand GUI and intuitive gameplay mechanics.

## Game Rules

### Set up

Game begins with player HQs as far apart on the map as possible. Positions of HQs will depend on the number of players.

Every player will start with 1000 funds

Every other building on the map will start as neutral and can be captured by infantry or mechs during the game

### Turns

At the start of each turn, the player will receive a base amount of income with the addition of further income depending on the number of buildings captured: each building contributing a 1000 funds

* + - 1. Units can be purchased at HQs or captured factories

The player will be able to purchase units with their funds

Units that are purchased will not be able to be used until the next turn

If the player has insufficient funds, they will not be able to purchase certain units

Units that are produced will not be able to perform any actions until the next turn

|  |  |
| --- | --- |
| Unit | Cost |
| Infantry | 1000 |
| Mech | 3000 |
| Tank | 7000 |
| APC | 5000 |

*Table of cost of units*

* + - 1. Units can move

Units’ movement is affected by the terrain around it and is affected by the number of movement points that they have

Moving a unit requires movement points (e.g. moving a unit over a mountain will cost more points compared to moving a unit along a road)

Moving a unit consumes fuel and once a unit has run out of fuel, it will no longer be able to be moved until it is resupplied by an APC or a Building

Units cannot move onto a tile that is already occupied by another unit

Vehicles are unable to move onto mountain tiles

|  |  |  |
| --- | --- | --- |
| Tile | Movement points required | |
| Foot | Treads |
| Plains | 1 | 2 |
| Road | 1 | 1 |
| Forest | 1 | 2 |
| Mountain | 2 | - |
| City | 2 | 3 |
| Factory | 2 | 3 |
| HQ | 2 | 3 |

*Table of relationship between tiles and movement points required when moving units*

|  |  |
| --- | --- |
| Unit | Movement points available |
| Infantry | 3 |
| Mech | 2 |
| Tank | 6 |
| APC | 6 |

*Table to show the number of movement points available to each unit type at the start of every turn*

* + - 1. Units can attack enemy units in adjacent tiles

The damage dealt to enemy units is determined by base damage, enemy defence, and the attacking unit’s health

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Attacking Unit | | | | |
| Defending Unit |  | Infantry | Mech | Tank | APC |
| Infantry | 55% | 65% | 75% | X |
| Mech | 45% | 55% | 70% | X |
| Tank | 5% | 55% | 55% | X |
| APC | 14% | 75% | 100% | X |

*Damage table for attacking units*

Attacking another unit will disable the attacking unit from performing any actions until the next turn

Attacking consumes ammo and once a unit has run out of ammo, it will no longer be able to attack until it is resupplied by an APC or a Building

* + - 1. Units can capture buildings

Non – vehicle units can capture buildings

Capturing buildings will disable a unit from performing any actions until the next turn

* + - 1. Defending Units

Units that are being attacked will have varying resistances to damage depending on the tile that they are situated on

This defence attribute will reduce the damage taken by a percentage of the base attack

|  |  |
| --- | --- |
| Tile | Damage prevented |
| Plains | 10% |
| Road | 0% |
| Forest | 20% |
| Mountain | 40% |
| City | 20% |
| Factory | 20% |
| HQ | 20% |

*Table of resistance to damage depending on defending unit’s tile*

* + - 1. Defending units can also counterattack. Given the following conditions are met:

The defending unit survives the attack

The defending unit can attack the attacking unit (e.g. unit has sufficient ammo)

* + - 1. The game will end once all HQs belong to a player or an AI

## Requirements of the end user

The following requirements are based on the interview with Dylan Canavan

### Starting the game

Upon starting the game, the user should be greeted with a menu screen displaying the different options to start the game – new game, load game and quit.

### Map

The map, number of players and options for whether one of the players is an AI should be displayed during game creation.

The map should be fairly balanced between the players so that one player does not have too great of an advantage over another player due to terrain generation.

### Players

The number of players should be variable between 2 – 4. Each player should be able to produce and control their units on the game map.

Producing units requires funds which should clearly be displayed to the user. Dylan also wanted AI turns to be short so that games required less time to complete.

### Artificial Intelligence (AI)

The AI should be of intermediate level difficulty so that the player will have enough of a challenge to not be bored but the game isn’t so difficult that the player will give up after a short amount of time.

Dylan also mentioned that the AI shouldn’t be able to do anything that the player isn’t able to do.

### Graphical User Interface (GUI)

Dylan wants the GUI of the game to be as simple as possible so that instructions are not required to know how to play.

While playing the game, the information required for the user should be displayed clearly: funds, turn number, current playing team. Dylan also said that having animations for units and tiles during the game would make the game more entertaining to play – This is not a priority as creating the frame for the game is more important.

## Research

Getting familiar with the game rules

I used the official Nintendo Advance Wars game manual1 as well as the Advanced Wars By Web Wiki Fandom2 to help me understand all the game rules before designing my documented design and UI. I am creating a simplified version of the game so not all rules from the original game will be necessary.

### Framework

#### Choosing the framework

I decided to use the Monogame framework as it’s highly optimised for game development especially when handling inputs and rendering graphics.

Although Monogame has many features and a steep learning curve, I believed that it was a better alternative to Windows Forms.

#### Getting familiar with the framework

To familiarise myself with the Monogame framework, I watched several tutorials and read a few guides around the inner workings of how the framework functioned.

YetiByte’s tutorial series3 on recreating the Google Dinosaur Game was particularly helpful in explaining how the system worked and I drew inspiration from their videos later when designing my game (1.7.1).

### Other Implementations

#### Example

Command Wars by Robostrum4

This implementation is written primarily in C++, whereas my code is written in C#. However, C++ syntax is similar enough to C#, so I didn’t have too much of an issue when interpreting the code.

The UI is written in JavaScript, but since I am looking at the algorithms, I won’t need to worry about this.

Advantages:

* Variables are named appropriately which makes the code easier to follow
* Various AI difficulties which can help with prototyping my own AI

Disadvantages:

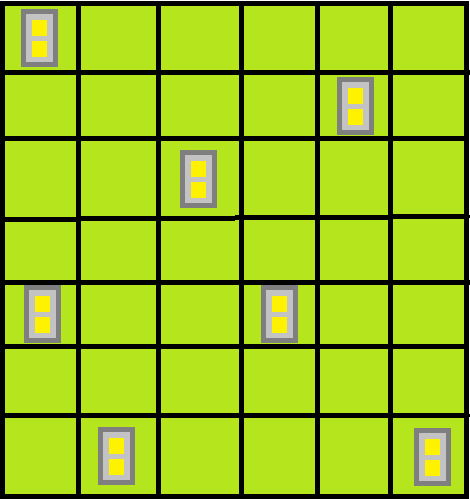
* Comments are limited making it harder to understand



### Algorithms

#### Procedural Generation

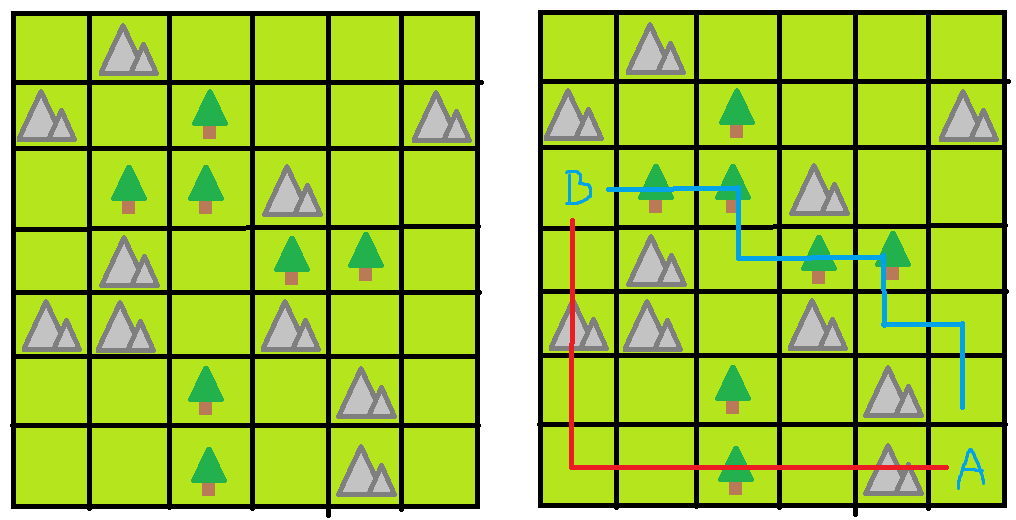
When the map is generated, buildings will need to be distributed throughout the map to not give one team an unfair advantage against another team.



Searching online I found that a potential algorithm for this would be Poisson Disc Sampling where there was a short paperby Robert Bridson on ‘Fast Poisson Disc Sampling in Arbitrary Dimensions’5 explaining how the algorithm functioned and a YouTube video by Sebastian Lague showing an implementation of this in Unity6.

#### Shortest path

The AI will be required to move its units across the map to attack the player. Since the map has multiple factors that affect and restrict movement, this will need to be considered when moving.



For example, here we have two of many possible paths. However, one path requires less movement points to traverse and therefore is the most efficient path.

As each tile will have a weight according to the number of movement points required to traverse it, a suitable algorithm would be Dijkstra’s algorithm or A\*.

The Wikipedia page for Dijkstra’s algorithm contains a pseudocode implementation of Dijkstra7 which I can adapt for my own usage.

#### Storing and Retrieving Game Data

A very large amount of data needs to be stored and retrieved. The procedurally generated map; all current active units in the game: type, location, attributes, etc; player data including AI: funds, income, owned buildings, and general game data such as which player’s turn it is, the turn number and the current game state to be used for the game logic

The use of XML or JSON can be used here. The use of the XML system is well documented8 which will help with learning how to implement it into my program.

1<https://fs-prod-cdn.nintendo-europe.com/media/downloads/games_8/emanuals/game_boy_advance_8/Manual_GameBoyAdvance_AdvanceWars_EN_DE_FR_ES_IT.pdf>

2<https://awbw.fandom.com/wiki/Advance_Wars_By_Web_Wiki>

3<https://www.youtube.com/watch?v=DJCQVJ83J1U&list=PLG2i_rSSIXXoFdh3WCDtGumklyIXt4KiY>

4<https://github.com/Robosturm/Commander_Wars>

5<https://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf>

6<https://youtu.be/7WcmyxyFO7o>

7<https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>

8<https://learn.microsoft.com/en-us/dotnet/api/system.xml?view=net-7.0>

## Objectives

### Menus

* + - 1. The main menu of the game is clear and easy to understand with defined options on proceeding to the game
      2. The main menu should display 3 options: New game, Load game and Quit
      3. Within the game the GUI should clearly display all the information the player will need

Current Player

Current balance

Turn number

End turn button

* + - 1. In-game menu should display 3 options: Resume, Save game, Quit game

### New Game

* + - 1. Game options for user to configure

The player can continue randomising the map

Number of players (2 – 4)

Whether a player should be an AI

* + - 1. All players will start with a balance of 1000

### Game Map

* + - 1. The map will be randomly generated
      2. Structures on the map should be distributed around the map and not focused in a particular area
      3. Roads should run between some of the structures in the map
      4. HQs on the map should be spread as far apart as possible
      5. All buildings will initially be neutral other than player HQs

### Save/Load game

* + - 1. Game will be saved into an XML or JSON file
      2. Saves:

Map

Structures on the map and their statistics (e.g team, defence bonus)

Game state

All unit positions and statistics (e.g health, team, remaining actions etc)

Which player’s turn it is

Whether a player is an AI

Turn number

* + - 1. Saved data is all loaded so that the game can be continued from where it was left

### Players

* + - 1. Will have a Boolean value determining whether they are an AI
      2. Start the game with an HQ tile where units can be produced
      3. Players that no longer have an HQ (due to capture) are taken out of the game unless they own another HQ
      4. All players in the game must be contained within a list to loop through whenever a player ends their turn
      5. All players should have a unique colour to identify them and any buildings they own

### Artificial Intelligence (AI)

* + - 1. The AI will be able to attack, manoeuvre and create units as well as capturing buildings
         1. The AI should try to attack the player when it is in an advantageous position
         2. The AI should attack units that are in range of its own units
         3. If a building can be captured, the AI should attempt to capture it
         4. The AI should attempt to create units at all possible production locations

### Attacking

* + - 1. When attacking, the user should be able to view the statistics of both the friendly attacking unit and the enemy defending unit
      2. The defending unit will lose health from taking damage
      3. After attacking the attacking unit will take damage from the defending unit as a counterattack
      4. After attacking, the unit will become immobile till the start of a new turn
      5. To indicate which unit the player is attacking, the enemy unit should be highlighted

### Tiles

* + - 1. Different tiles will have different attributes
      2. Roads

Improved mobility for vehicles

Units will have no defence bonus

* + - 1. Plains

Mobility of vehicles is lowered

* + - 1. Mountain

Units on a mountain will have a greater defence bonus

Non vehicles can traverse mountains

Moving onto a mountain reduces mobility

* + - 1. Buildings – can be captured, produces 1000 funds per turn and can resupply units
      2. HQ

Can produce units

* + - 1. Cities

Units will take reduced damage while on this tile

* + - 1. Factories

Produces units

### Selecting/Manoeuvring units

* + - 1. Units can only be moved once per turn
      2. Only one unit is able to be selected at any one time
      3. Units are unable to move onto a tile where a unit is already present
      4. Newly created units are unable to move for a turn
      5. Units have a limited amount of fuel and must be resupplied when they have run out to move again
      6. The unit selected shows its stats in a small list
      7. When moving a selected unit, the possible tiles that it can move to should be highlighted to the player

### Units

* + - 1. Different units will have different attributes
      2. A unit’s team should be identifiable by the colour of the unit
      3. Infantry

Basic starting unit with well-rounded stats

Can capture buildings

* + - 1. Mech

Greater damage against vehicles

Can capture buildings

* + - 1. APC

Ability to resupply adjacent units

Ability to carry other units that are not vehicles

Cannot capture buildings

* + - 1. Tank

High Damage

High armour

Low mobility and fuel

Expensive

Cannot capture buildings

* + - 1. All units will have:

Health

Ammo

Fuel

Defence

Mobility distance

### Game

* + - 1. Win condition of game is when there is only 1 remaining HQ left
      2. Units can be produced on HQ and factory tiles given there is not already a unit there
      3. When a turn has ended, all unit states are reset
      4. When the game ends a Game over screen will appear and options to return to the menu or quit game will be displayed
      5. Animations play throughout the game when units are moving, attacking or capturing

## Prototyping and Modelling

### Simple Class Diagram

Diagram, engineering drawing

Description automatically generated

### Storing and retrieving game data

Using the information from my research in 1.5.3.3, I created an implementation in a simple console application for storing data in an XML file. Which produced the following file containing the data of 10 units:

<?xml version="1.0" encoding="utf-8"?>

<ArrayOfUnitData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<UnitData>

<Health>0</Health>

<Ammo>0</Ammo>

<Fuel>0</Fuel>

<Defence>0</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>1</Health>

<Ammo>5</Ammo>

<Fuel>20</Fuel>

<Defence>30</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>2</Health>

<Ammo>10</Ammo>

<Fuel>40</Fuel>

<Defence>60</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>3</Health>

<Ammo>15</Ammo>

<Fuel>60</Fuel>

<Defence>90</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>4</Health>

<Ammo>20</Ammo>

<Fuel>80</Fuel>

<Defence>120</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>5</Health>

<Ammo>25</Ammo>

<Fuel>100</Fuel>

<Defence>150</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>6</Health>

<Ammo>30</Ammo>

<Fuel>120</Fuel>

<Defence>180</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>7</Health>

<Ammo>35</Ammo>

<Fuel>140</Fuel>

<Defence>210</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>8</Health>

<Ammo>40</Ammo>

<Fuel>160</Fuel>

<Defence>240</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<UnitData>

<Health>9</Health>

<Ammo>45</Ammo>

<Fuel>180</Fuel>

<Defence>270</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

</ArrayOfUnitData>

The values for the attributes of the units stored above are arbitrary and don’t reflect the realistic values of a unit’s attributes in the game.

# Documented Design

## Overview

Each section of the design will have a link back to the relevant objectives in 1.6

Descriptions of algorithms will be added above if deemed necessary and any extra methods will be described after the pseudocode.

## Algorithms

Note that the design and algorithms used for the AI are not here and instead has its own dedicated section 2.7.

### Common Variables

The following table is a list of commonly used variables in the key algorithms identified.

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Map | A 2D array containing the tiles that the map is composed of |
| Units | A list of all the units in the game |
| CurrentUnit | The current unit that has been selected by the player |
| CurrentTile | The current tile that has been selected by the player |
| Unit.Type | The type of the unit: Infantry, Mech, Tank, APC |
| Unit.State | The state of the unit: None, Moved, Used, Dead |
| Tile.Type | The type of the tile: None, Plains, Forest, Mountain, Road, City, Factory, HQ |
| Texture | The sprite sheet containing all the game assets |
| ReachableTiles | The tiles that a unit can move to from its current position |
| AttackableTiles | The tiles where an enemy unit is located on that the current unit is able to attack |

### Creating Unit Sprites

|  |  |
| --- | --- |
| **Description:**  The following algorithm from the Unit class is used to generate a unit sprite. The shift values are used to change what sprite should be selected from the sprite sheet. The Team of the unit is passed into the constructor of the Unit object upon creation. | |
| **Input:**  unitType – the type of unit being created | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE CreateUnitSprite(unitType)  teamShift = (Team) \* SPRITE\_SHEET\_TEAM\_SHIFT  unitShift = (unitType - 1) \* SPRITE\_SHEET\_UNIT\_SHIFT  unitSprite = new Sprite(Texture, X\_SPRITE\_SHEET\_START\_POS + teamShift, Y\_SPRITE\_SHEET\_START\_POS + unitShift, UNIT\_WIDTH, UNIT\_HEIGHT)  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.10.2** | |

### Creating Tile Sprites

|  |  |
| --- | --- |
| **Description:** The following algorithm is used to generate the tile sprite. There are two different methods for generating tile sprites. One which is used to manually create sprites with the row and column in the sprite sheet. The other which creates the sprite depending on the type of the tile and utilises the first method to do so. | |
| **Input:**  TileColumn – The column that the sprite is located on in the sprite sheet from the starting location  TileRow - The row that the sprite is located on in the sprite sheet from the starting location | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE CreateTileSprite(TileColumn = 0, TileRow = 0)  TileColumn = TileColumn \* TILE\_WIDTH  TileRow = TileRow \* TILE\_HEIGHT  TileSprite = new Sprite(Texture, X\_SPRITE\_SHEET\_START\_POS + TileColumn, Y\_SPRITE\_SHEET\_START\_POS + TileRow, TILE\_WIDTH, TILE\_HEIGHT)  SetTileAttributes()  ENDSUBROUTINE  SUBROUTINE CreateTileSpriteOnType()  SWITCH (Type)  CASE TileType.Plains:  CreateTileSprite()  BREAK  CASE TileType.Forest:  CreateTileSprite(1, 0)  BREAK  CASE TileType.Mountain:  CreateTileSprite(2, 0)  BREAK  CASE TileType.City:  CreateTileSprite(-5 + Team, 0)  BREAK  CASE TileType.Factory:  CreateTileSprite(-5 + Team, 1)  BREAK  CASE TileType.HQ:  CreateTileSprite(-5 + Team, 2)  BREAK  ENDSWITCH  ENDSUBROUTINE | |
| **Relevant Objective:** | |
| **Extra:**  The SetTileAtttribute method sets the attributes of the tile depending on its type | |

### Generating random points

|  |  |
| --- | --- |
| **Description:**  The following is an algorithm for Poisson Disc Sampling that generates a set of random points where no two points can be closer than the specified radius. This algorithm is used during map generation when adding structures (2.2.7) | |
| **Input:**  radius – specified set distance between generated points  mapSize – Region to produce points in | **Output:** points – List of point coordinates distributed within the mapSize with a set distance radius between them |
| **Pseudocode:**  SUBROUTINE GetPoints(radius, mapSize, k = 30)  cellSize = radius / sqrt(2)  Width = int(mapSize.X / cellSize) + 1  Height = int(mapSize.Y / cellSize) + 1  grid = new int[Width, Height]  points = new List<Vector2>()  startPoints = new List<Vector2>()  initialPoint = new Vector2(RANDOM(mapSize.X), RANDOM(mapSize.Y))  startPoints.Add(initialPoint)  WHILE startPoints is not empty  randomIndex = RANDOM(startPoints.Count)  startPoint = startPoints[randomIndex]  potentialPointFound = false  FOR tries = 0 to k  angle = random.NextDouble() \* 2 \* pi  direction = new Vector2(SIN(angle), COS(angle))  candidatePoint = startPoint + direction \* RANDOM (radius, radius \* 2)  IF IsValid(candidatePoint, mapSize, cellSize, points, grid, radius)  points.Add(candidatePoint)  startPoints.Add(candidatePoint)  grid[int(candidatePoint.X / cellSize), int(candidatePoint.Y / cellSize)] = points.Count  potentialPointFound = true  BREAK  ENDIF  ENDFOR  IF potentialPointFound is FALSE  startPoints.RemoveAt(randomIndex)  ENDIF  ENDWHILE  RETURN points  ENDSUBROUTINE  SUBROUTINE IsValid(potentialPoint, mapSize, cellSize, points, grid, radius)  IF potentialPoint.X >= 0 and potentialPoint.X < mapSize.X and potentialPoint.Y >= 0 and potentialPoint.Y < mapSize.Y  cell = new Vector2(int(potentialPoint.X / cellSize), int(potentialPoint.Y / cellSize))  startX = max(0, cell.X - 2)  endX = min(cell.X + 2, grid.GetLength(0) - 1)  startY = max(0, cell.Y - 2)  endY = min(cell.Y + 2, grid.GetLength(1) - 1)  FOR x = startX to endX  FOR y = startY to endY  pointIndex = grid[x, y] - 1  IF pointIndex != -1  squrdDistance = SquaredDistance(potentialPoint, points[pointIndex])  IF squrdDistance < radius^2  RETURN false  ENDIF  ENDIF  ENDFOR  ENDFOR  RETURN true  ENDIF  RETURN false  ENDSUBROUTINE  SUBROUTINE SquaredDistance(start, end)  x = start.X - end.X  y = start.Y - end.Y  return x^2 + y^2  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.3.2** | |

### Getting Adjacent Tiles

|  |  |
| --- | --- |
| **Input:**  tile – the starting tile that the algorithm will get the adjacent tiles of | **Output:**  neighbours – the list of tiles that are adjacent to the starting tile |
| **Pseudocode:**  SUBROUTINE GetNeighbours(tile)  neighbors = new List<Tile>()  X = tile.MapGridPos.X  Y = tile.MapGridPos.Y  IF X > 0  neighbors.Add(Map[X - 1, Y])  ENDIF  IF X < Map.GetLength(0) - 1  neighbors.Add(Map[X + 1, Y])  ENDIF  IF Y > 0 THEN  neighbors.Add(Map[X, Y - 1])  ENDIF  IF Y < Map.GetLength(1) - 1  neighbors.Add(Map[X, Y + 1])  ENDIF  RETURN neighbors  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.7, 1.6.10.5** | |

### Map Generation

#### Base Map Generation

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE GenerateBaseMap():  x = Position.X  y = Position.Y  FOR i = 0 to MapHeight - 1  x = Position.X  FOR j = 0 to MapWidth - 1  randomTile = RandomTile()  tempPosition = Vector2(x, y)  newTile = Tile(tempPosition, Texture)  newTile.MapGridPos = Vector2(j, i)  SWITCH (randomTile):  CASE 0:  newTile.Type = TileType.Plains  BREAK  CASE 1:  newTile.Type = TileType.Forest  BREAK  CASE 2:  newTile.Type = TileType.Mountain  BREAK  ENDSWITCH  Map[j, i] = newTile  x = x + 56  ENDFOR  y = y + 56  ENDFOR  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.3.1** | |

#### Adding Structures to Map

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE GenerateStructure(StructureType)  Type = None  points = empty list  IF StructureType == "City"  Type = City  ELSEIF StructureType == "Factory"  Type = Factory  StructureSparsity \*= 3  ENDIF  points = PoissonDiscSampling.GetPoints(StructureSparsity, MapSize)  FOREACH point in points  newGridX = (int)(point.X) / TILE\_DIMENSIONS  newGridY = (int)(point.Y) / TILE\_DIMENSIONS  newGridPos = new Vector2(Map[newGridX, newGridY].Position.X, Map[newGridX, newGridY].Position.Y)  newStructure = new Tile(newGridPos, Texture)  IF Map[newGridX, newGridY].Type != City  newStructure.MapGridPos = new Vector2(newGridX, newGridY)  Map[newGridX, newGridY] = newStructure  Map[newGridX, newGridY].Type = Type  Structures.add(newStructure)  ENDIF  ENDFOREACH  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.3.2, 1.6.3.5** | |

#### Generating Roads Between Structures

|  |  |
| --- | --- |
| **Description:** The following algorithm consists of three methods which work together to generate the road tiles in the map. This is done to reduce repeated code. | |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE GenerateRoads()  firstStructureGridPos  nextStructureGridPos  FOR i to Structures.Count – 1 with i += 2  firstStructureGridPos = Structures[i].MapGridPos  nextStructureGridPos = Structures[i + 1].MapGridPos  BuildRoad(firstStructureGridPos, nextStructureGridPos)  ENDFOR  ENDSUBROUTINE  SUBROUTINE BuildRoad(firstStructurePos, nextStructurePos)  x0 = (int)firstStructurePos.X  y0 = (int)firstStructurePos.Y  x1 = (int)nextStructurePos.X  y1 = (int)nextStructurePos.Y  WHILE x0 != x1  IF x0 > x1  x0 = x0 - 1  CreateRoadTile(x0, y0, 3)  ELSEIF x0 < x1  x0 = x0 + 1  CreateRoadTile(x0, y0, 3)  ENDIF  ENDWHILE  WHILE y0 != y1  IF y0 > y1  y0 = y0 - 1  CreateRoadTile(x0, y0, 4)  ELSEIF y0 < y1  y0 = y0 + 1  CreateRoadTile(x0, y0, 4)  ENDIF  ENDWHILE  ENDSUBROUTINE  SUBROUTINE CreateRoadTile(X, Y, direction)  IF Map[X, Y].Type != TileType.City AND  Map[X, Y].Type != TileType.Factory AND  Map[X, Y].Type != TileType.Mountain AND  Map[X, Y].Type != TileType.HQ  Tile roadTile = new(Map[X, Y].Position, Texture)  roadTile.Type = TileType.Road  roadTile.MapGridPos = new Vector2(X, Y)  Map[X, Y] = roadTile  Map[X, Y].CreateTileSprite(direction)  ENDIF  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.3.3** | |

### Dijkstra Pathfinding

|  |  |
| --- | --- |
| **Input:**  startingTile – The tile that the search starts from  unit – The unit that is moving | **Output:**  reachableTiles – the tiles that the unit can move to |
| **Pseudocode:**  SUBROUTINE FindReachableTiles(startingTile, unit)  List<Tile> reachableTiles  PriorityQueue<Tile, int> queue  startingTile.TotalCost = 0  queue.Enqueue(startingTile, 0)  WHILE queue.Count > 0  Tile currentTile = queue.Dequeue()  reachableTiles.Add(currentTile)  FOREACH neighbor IN mapManager.GetNeighbours(currentTile)  int cost = currentTile.TotalCost + mapManager.GetCost(currentTile, unit)  IF NOT reachableTiles.Contains(neighbor) AND cost <= unit.MovementPoints  neighbor.TotalCost = cost  queue.Enqueue(neighbor, cost)  ENDIF  ENDFOR  ENDWHILE  RETURN reachableTiles  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.9** | |
| **Extra:**  The mapManager.GetCost method simply returns an integer of the movementPointCost of moving the unit on the specified tile | |

### Player Actions

The following algorithms are Player actions

The parameter gameTime would have been used for animations but unfortunately this was not complete (See Evaluation 5.3)

Calls to \_gameUI are primarily used to display the GUI to the player.

Calls to \_inputController are used to get the Selected Unit/Tile, find the tile that a Unit is located or find the Unit that is located on a tile (3.5.2 Lines 97- 120 and 135 – 159)

#### Player Select

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE PlayerSelect(gameTime)  ProcessButtonsOnly = false  UpdateUnitStats()  \_gameUI.ClearAttackableOverlay()  IF SelectedUnit != null  \_gameUI.DisplayAttributes(SelectedUnit)  IF SelectedUnit.State != UnitState.Used AND SelectedUnit.State != UnitState.Moved AND SelectedUnit.State != UnitState.Dead  SelectedUnit.State = UnitState.None  ENDIF  SelectedUnit.Selected = false  ENDIF  SelectedUnit = \_inputController.GetSelectedUnit()  SelectedTile = \_inputController.GetSelectedTile()  IF SelectedUnit != null  \_gameUI.ChangeSelectedPosition(SelectedUnit.Position)  \_gameUI.DrawSelectedUI = true  SelectedUnit.Selected = true  \_gameUI.DisplayAttributes(SelectedUnit)  IF SelectedUnit.Team == CurrentPlayer.Team AND SelectedUnit.State != UnitState.Used  gameState = GameState.SelectAction  ELSE  SelectedUnit.Selected = false  ENDIF  ENDIF  IF SelectedTile != null  \_gameUI.ChangeSelectedPosition(SelectedTile.Position)  \_gameUI.DrawSelectedUI = true  \_gameUI.DisplayAttributes(null, SelectedTile)  IF (SelectedTile.Type == TileType.Factory OR SelectedTile.Type == TileType.HQ) AND SelectedTile.Team == CurrentPlayer.Team  ProcessButtonsOnly = true  gameState = GameState.PlayerProduceUnit  ENDIF  ENDIF  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.9.2** | |
| **Extra:**  UpdateUnitStats() updates the defence values of all units depending on the tile they are situated on | |

#### Player Movement

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE PlayerMove(GameTime gameTime)  IF reachableTiles.Count != 0  WHILE gameState == GameState.PlayerMove AND DrawRan  DrawRan = false  FOREACH (Tile tile IN reachableTiles  IF \_inputController.MouseCollider.Intersects(tile.Collider)  AND \_inputController.LeftMouseClicked()  AND SelectedUnit.Fuel > 0    SelectedUnit.Position = tile.Position  SelectedUnit.Fuel--  SelectedUnit.State = UnitState.Moved  \_gameUI.ChangeSelectedPosition(SelectedUnit.Position)  gameState = GameState.SelectAction  ENDIF  ENDFOREACH  IF (SelectedUnit.Fuel <= 0)  gameState = GameState.SelectAction  ENDIF  ENDWHILE  ELSE  gameState = GameState.PlayerSelect  ENDIF  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.9.1, 1.6.9.5** | |

#### Player Attacking

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE PlayerAttack(GameTime gameTime)  IF SelectedUnit.State != UnitState.Used  AND SelectedUnit.Type != UnitType.APC  AND attackableTiles.Count != 0  WHILE gameState == GameState.PlayerAttack AND DrawRan  DrawRan = false  FOREACH Tile tile IN attackableTiles  IF \_inputController.MouseCollider.Intersects(tile.Collider)  AND \_inputController.LeftMouseClicked()  AND SelectedUnit.Ammo > 0    defendingUnit = \_inputController.GetTileUnit(tile)  defendingUnit.Health -= \_gameUI.CalculateDamage(SelectedUnit, defendingUnit)  IF defendingUnit.Health > 0 AND defendingUnit.Ammo > 0    SelectedUnit.Health -= \_gameUI.CalculateDamage(defendingUnit, SelectedUnit)  defendingUnit.Ammo—  ENDIF  SelectedUnit.State = UnitState.Used  gameState = GameState.PlayerSelect  ENDIF  IF SelectedUnit.Ammo <= 0  gameState = GameState.SelectAction  ENDIF  ENDFOR  ENDWHILE  ENDIF  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.7.2** | |
| **Extra:**  \_gameUI.CalculateDamage takes as input the attacking unit and the defending unit. It then returns an integer value as the damage that has been dealt (3.3.9 Lines 671-682) | |

#### Player Producing Units

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE PlayerProduceUnit(GameTime gameTime)  WHILE gameState == GameState.PlayerProduceUnit AND DrawRan  DrawRan = false  unitType = \_gameUI.ProcessUnitProduction(gameTime, PressedButton)  IF unitType == -1  gameState = GameState.PlayerSelect  ENDIF  IF unitType != -1 AND unitType != -2  Unit newUnit = new(InGameAssets, SelectedTile.Position, unitType, CurrentPlayer.Team)  IF CurrentPlayer.Funds >= newUnit.CostToProduce  CurrentPlayer.Funds -= newUnit.CostToProduce  newUnit.State = UnitState.Used  \_unitManager.AddUnit(newUnit)  gameState = GameState.PlayerSelect  ENDIF  ENDIF  ENDWHILE  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.9.4, 1.6.11.2** | |

#### Player Resupplying Units

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE PlayerResupply(GameTime gameTime)  IF SelectedUnit.Type == UnitType.APC  SelectedUnit.State = UnitState.Used  List<Tile> neighbours = \_gameMap.GetNeighbours(CurrentUnitTile)  FOREACH Tile neighbour IN neighbours  Unit neighbouringUnit = \_inputController.GetTileUnit(neighbour)  IF neighbouringUnit != null AND neighbouringUnit.Team == SelectedUnit.Team  neighbouringUnit.RefreshUnitAttributes()  ENDIF  ENDFOR  IF (CurrentUnitTile.Type == TileType.City  OR CurrentUnitTile.Type == TileType.Factory  OR CurrentUnitTile.Type == TileType.HQ)  AND CurrentUnitTile.Team == SelectedUnit.Team    SelectedUnit.RefreshUnitAttributes()  ENDIF  gameState = GameState.PlayerSelect  ELSE  SelectedUnit.State = UnitState.Used  SelectedUnit.RefreshUnitAttributes()  gameState = GameState.PlayerSelect  ENDIF  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.8.5, 1.6.10.5,** | |
| **Extra:**  The RefreshUnitAttribute method simply sets the ammo and fuel of units back to their maximum value (3.3.7 Lines 83 – 119) | |

#### Player Capturing Structures

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE PlayerCapture(gameTime)  CurrentUnitTile.Team = SelectedUnit.Team  SelectedUnit.State = UnitState.Used  CurrentUnitTile.CreateTileSpriteOnType()  GameOver = CheckWinner(CurrentPlayer.Team)  CheckHQ()  IF GameOver  menuState = MenuState.GameOver  ENDIF  gameState = GameState.PlayerSelect  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.8.5, 1.6.10** | |
| **Extra:**  The CheckWinner method checks for whether there are enemy HQs left, if there aren’t any then the gameState is set to GameOver (3.1 Lines 635 – 645). The CheckHQ method checks for whether any of the players have lost their HQ. If they have then the player is removed from the game along with their units (3.1 Lines 647 – 703). | |

### Saving/Loading Games

The classes and methods used during loading and saving can all be found within 3.5:  
.5, .6, .7, .8, .9, .10

#### Saving

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE SaveGame(gameTime)  unitData = new List<UnitData>  mapData = new List<TileData>  structuresData = new List<TileData>  playerData = new List<PlayerData>  gameStateData = new GameStateData(TurnNumber, CurrentPlayerIndex, AddAI)  ComputerData = null  IF AddAI  ComputerData = new AIData(Computer)  ENDIF  FOR unit IN \_unitManager.units  unitData.Add(new UnitData(unit))  ENDFOR  FOR tile IN \_gameMap.Map  mapData.Add(new TileData(tile))  ENDFOR  FOREACH tile IN \_gameMap.Structures  IF tile.Type != TileType.HQ  structuresData.Add(new TileData(tile))  ENDIF  ENDFOR  FOR player IN Players  playerData.Add(new PlayerData(player))  ENDFOR  IF AddAI  gameData = new GameData(unitData, mapData, structuresData, playerData, gameStateData, \_gameMap.MapWidth, \_gameMap.MapHeight, ComputerData)  ELSE  gameData = new GameData(unitData, mapData, structuresData, playerData, gameStateData, \_gameMap.MapWidth, \_gameMap.MapHeight)  ENDIF  serializer = new XmlSerializer(typeof(GameData))  USING streamWriter = new StreamWriter(SAVE\_GAME\_PATH)  serializer.Serialize(streamWriter, gameData)  ENDUSING  menuState = MenuState.PlayingGame  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.4** | |

#### Loading

|  |  |
| --- | --- |
| **Input:**  N/A | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE LoadGame(GameTime gameTime)  IF File.Exists(SAVE\_GAME\_PATH)  \_unitManager.DrawUnits = true;  \_gameMap.DrawMap = true;  \_unitManager.ClearUnits();  Players.Clear();  \_gameMap.ClearMap();  \_gameMap.Structures.Clear();  \_gameMap.HQs.Clear();  GameData gameData;    Map = new()  XmlSerializer serializer = new(typeof(GameData));  USING (StreamReader streamReader = new(SAVE\_GAME\_PATH))  gameData = (GameData)serializer.Deserialize(streamReader);  ENDUSING  FOREACH unitData IN gameData.Units  \_unitManager.AddUnit(unitData.FromUnitData(InGameAssets));  ENDFOR  FOR EACH tileData IN gameData.Map  Map.Add(tileData.FromTileData(InGameAssets));  ENDFOR  FOR EACH playerData IN gameData.Players O  Players.Add(playerData.FromPlayerData());  ENDFOR  FOREACH structure IN gameData.Structures  \_gameMap.Structures.Add(structure.FromTileData(InGameAssets));  ENDFOR  \_gameMap.MapWidth = gameData.MapWidth;  \_gameMap.MapHeight = gameData.MapHeight;  \_gameMap.ResetMapSize();  FOREACH tile IN Map  \_gameMap.Map[(int)tile.MapGridPos.X, (int)tile.MapGridPos.Y] = tile;  IF tile.Type == TileType.HQ  \_gameMap.HQs.Add(tile);  ENDIF  ENDFOR  \_gameMap.RegenerateMap();  AddAI = gameData.GameStateData.AddAI;  CurrentPlayerIndex = gameData.GameStateData.CurrentPlayerIndex;  TurnNumber = gameData.GameStateData.TurnNumber;  CurrentPlayer = Players[CurrentPlayerIndex];  IF CurrentPlayer.IsAI  Computer = gameData.Computer.FromAIData(InGameAssets, \_gameMap, \_unitManager, \_gameUI, \_inputController);  gameState = GameState.AITurn;  ENDIF  menuState = MenuState.PlayingGame;  ELSE  menuState = MenuState.Initial;  ENDIF  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.4** | |
| **Extra:**  The RegenerateMapMethod recreates all the roads and Tile sprites in the map (3.3.8 Lines 62 – 66) | |

### Player Income

|  |  |
| --- | --- |
| **Input:**  player – The player that is gaining the funds at the start of their turn | **Output:**  N/A |
| **Pseudocode:**  SUBROUTINE Income(player)  FOREACH Tile structure in \_gameMap.Structures  IF structure.Team == player.Team  player.Funds += 1000  ENDIF  ENDFOR  ENDSUBROUTINE | |
| **Relevant Objective: 1.6.8.5** | |

### Player Loop

|  |  |
| --- | --- |
| **Description:** How a player’s turn would look. Note that this is how a normal turn would generally look but player actions are not limited to this. | |
| **Input:**  N/A | **Output:**  N/A |
| **Flowchart:** | |
| **Relevant Objective: 1.6.11.1** | |

## Classes

This section contains class definitions for all the classes in the solution

### Graphics

#### Sprite

|  |
| --- |
| **Attributes:**  + Height : int  + Texture : Texture2D  + TintColour : Color  + X : int  + Y : int |
| **Methods:**  + Sprite()  + Draw() : void |

#### Font

|  |
| --- |
| **Attributes:**  + font : SpriteFont  + Text : stinrg  + TextMiddlePoint : Vector2  + TintColour : Color |
| **Methods:**  + Font()  + WriteText : void |

### Entities

#### EntityManager

|  |
| --- |
| **Attributes:**  - entities : List<IGameEntity>  - entitiesToRemove : List<IGameEntity> |
| **Methods:**  + AddEntity() : void  + ClearEntites: void  + Draw() : void  + RemoveEntity(): void  + Update(): void |

#### Button

|  |
| --- |
| **Attributes:**  - ButtonPosition : Vector2  - buttonSprite : Sprite  - Height: int  - spriteFont : SpriteFont  - text : Font  - TextPosition : Vector2  - Texture : Texture2D  - Width : int  - X\_SPRITE\_SHEET\_START\_POS : int  - Y\_SPRITE\_SHEET\_START\_POS : int  - ButtonShiftX : int  - ButtonShiftY : int  - ButtonType : int  + Collider : Rectangle  + DrawButton : bool  + DrawOrder : int  + ID : int  + Pressed : bool |
| **Methods:**  + Button()  + CentreText() : void  + CreateButtonSprite() : void  + Draw() : void  + GetButtonType() : void  + Update() : void  + UpdateButtonText() : void |

#### ButtonManager

|  |
| --- |
| **Attributes:**  + buttons : List<Button>  + ID : int  + DrawOrder : int |
| **Methods:**  + AddButton() : void  + AddButtons() : void  + Draw() : void  + DrawButtonIDs() : void  + Update() : void  + UpdateButtonText() : void |

#### Unit

|  |
| --- |
| **Attributes:**  - SPRITE\_SHEET\_TEAM\_SHIFT : int  - SPRITE\_SHEET\_UNIT\_SHIFT : int  - UNIT\_HEIGHT : int  - UNIT\_WIDHT : int  - unitSprite : Sprite  - X\_SPRITE\_SHEET\_START\_POS : int  - Y\_SPRITE\_SHEET\_START\_POS : int  + Ammo : int  + Collider : Rectangle  + CostToProduce : int  + Defence : float  + DrawOrder : int  + Fuel : int  + Health : int  + MovementPoints : int  + Position : Vector2  + Selected : bool  + State : UnitState  + Team : int  - Texture : Texture2D  + Type : UnitType  + UnitTypeInt : int |
| **Methods:**  + CreateUnitSprite() : void  + Draw() : void  + RefrehsUnitAttributes() : void  - SetStartingAttributes() : void  + Unit()  + Update() : void |

#### UnitManager

|  |
| --- |
| **Attributes:**  + units : List<Unit>  + unitsToRemove : List<Unit>  + DrawOrder : int  + DrawUnits : bool |
| **Methods:**  + AddUnit() : void  + ClearUnits(): void  + Draw(): void  + GetUnitPositions(): List<Vector2>  + RemoveUnit() : void  + ResetUnitStates() : void  + UnitManager()  + Update() : void |

#### Tile

|  |
| --- |
| **Attributes:**  - TILE\_HEIGHT : int  - TILE\_WIDHT : int  - X\_SPRITE\_SHEET\_START\_POS : int  - Y\_SPRITE\_SHEET\_START\_POS : int  + Collider : Rectangle  + DrawOrder : int  + MapGridPos : Vector2  + Position : Vector2  + Selected : bool  + Team : int  - Texture : Texture2D  + TileSprite : Sprite  + TotalCost : int  + Type : TileType |
| **Methods:**  + CreateTileSprite() : void  + CreateTileSpriteOnType() : void  + Draw() : void  + SetTileAttributes() : void  + Tile()  + Update() : void |

#### MapManager

|  |
| --- |
| **Attributes:**  - TILE\_DIMENSIONS : int  - WINDOW\_HEIGHT : int  - WINDOW\_WIDHT : int  + Collider : Rectangle  + DrawMap : bool  + DrawOrder : int  + HQs : List<Tile>  + Map : Tile[,]  + MapHeight : int  + MapSize : Vector2  + MapWidth : int  - NumOfPlayers : int  + Position : Vector2  + Structures : List<Tile>  - StructureSparsity : int  - Texture : Texture2D |
| **Methods:**  - BuildRoad() : void  + ClearMap() : void  - CreateHQTile() : void  - CreateRoadTile() : void  - CreateTileSprites() : void  + Draw() : void  - GenerateBaseMap() : void  + GenerateHQs() : void  - GenerateMap() : void  - GenerateRoads() : void  - GenerateStructure() : void  + GetCost() : int  + GetNeighbours : List<Tile>  + MapManager()  - RandomTile(): int  + RegenerateMap() : void  + ResetMapSize() : void  + Update() : void |

#### GameUI

|  |
| --- |
| **Attributes:**  - \_buttonManager : ButtonManager  - \_gameMap : MapManager  - \_pathFinder : Dijkstra - \_unitManager : UnitManager  - attackableOverlay : List<Tile>  - attackableTiles : List<Tile>  - baseDamageDictionary : Dictionary<ValueTuple<UnitType, UnitType>, int>  - CentreButtonX : int  - CurrentPlayer: Player  - CurrentTile : tile  - CurrentUnit : Unit  - DrawAttackable : bool  - DrawReachable : bool  - Font : SpriteFont  - moveableOverlay : List<Tile>  - numOfPlayers : int  - reachableTiles : List<Tile>  - SelectedUI : Tile  - Texture : Texture2D  - tilesToBeRemoved : List<Tile>  + DrawOrder : int  + DrawSelectedUI : bool  - BasicWarsTitle : Button  - NewGameButton : Button  - LoadGameButton : Button  - QuitGameButton : Button  - NumOfPlayersInfo : Button  - Players2Button : Button  - Players3Button : Button  - Players4Button : Button  - RefreshMapButton : Button  - StartGameButton : Button  - MenuButton : Button  - TurnNumberInfo : Button  - CurrentPlayerTeamInfo : Button  - CurrentPlayerFundsInfo : Button  - EndTurnButton : Button  - PauseGameButton : Button  - PlayerIdleButton : Button  - PlayerMoveButton : Button  - PlayerAttackButton : Button  - ReturnButton : Button  - CaptureButton : Button  - AttributeDisplayInfo : Button  - HealthInfo : Button  - AmmoInfo : Button  - FuelInfo : Button  - DefenceInfo : Button  - TypeInfo : Button  - UnitInfantryButton : Button  - UnitMechButton : Button  - UnitTankButton : Button  - UnitAPCButton : Button  - ResumeGameButton : Button  - SaveGameButton : Button  - MainMenuButton : Button  - GameOverInfo : Button  - WinnerInfo : Button  - IncreaseMapSizeButton : Button  - DecreaseMapSizeButton : Button  - UnitResupplyButton : Button  - AddAIInfo : Button  - AIPlayerTrue : Button  - AIPlayerFalse : Button |
| **Methods:**  + CalculateDamage() : int  + ChangeMap() : void  + ChangeSelectedPosition() : void  + ClearAttackableOverlay() : void  + ClearMoveableOverlay() : void  + DisplayAttributes() : void  + DispalyPlayerActions() : GameState  + Draw() : void  + GameOver() : MenuState  + GameUI()  + GetAttackableTiles() : List<Tile>  + GetPlayers() : List<Player>  + GetReachableTiles() : List<Tile> + Init() : MenuState  + InitialiseButtons() : void  + NewGame() : MenuState  + PausedGame() : MenuState  + ProcessUnitProduction() : int  + Turn() : GameState  + Update() : void |

#### Player

|  |
| --- |
| **Attributes:**  + Colour : string  + Funds : int  + HasHQ : bool  + IsAI : bool  + Team : int |
| **Methods:**  - GetTeam Colour() : void  + Player() |

### System

#### AI : Player

|  |
| --- |
| **Attributes:**  - AIUnits : List<Unit>  - closeByStructure : List<Tile>  - EnemyHQs : List<Tile>  - FriendlyBuildings : List<Tile>  - HQToAttack : Tile  - random : Random  - reachableTiles : List<Tile  - \_gameMap : MapManager  - \_gameUI : GameUI  - \_inputController : InputController  - \_unitManager : UnitManager  - HQ : tile  - State : AIState  - Texture : Texture2D |
| **Methods:**  + AI()  - AttackNeighbouringUnits() : void  - CaptureStructures() : void  - CloseByStructure() : bool  - EnemyCloseToHQ() : bool  - GetAIUnits() : void  - GetEnemyHQs() : void  - GetFriendlyBuildings() : void  - GetReachableTiles() : List<Tile>  - GetSquaredDistance() : float  - MoveToRandomReachable() : void  - MoveTowardsTile() : void  - ProduceUnit() : void  + RefreshAI() : void  + RunAILogic() : void  - Winning() : bool |

#### InputController

|  |
| --- |
| **Attributes:**  - \_buttonManager : ButtonManager  - \_gameMap : MapManager  - \_unitManager : UnitManager  - currentMouseState : MouseState  - previousMouseState : MouseState  + MouseCollider : Rectangle |
| **Methods:**  + ChangeMap() : void  + GetButtonPressed() : Button  + GetSelectedTile() : Tile  + GetSelectedUnit() : Unit  + GetTileUnit() : Unit  + GetUnitTIle() : Tile  + InputController()  + LeftMouseClicked() : bool  + ProcessControls() : void  + UpdateMouseState() : void |

#### Dijkstra

|  |
| --- |
| **Attributes:**  - mapManager : MapManager |
| **Methods:**  + Dijkstra  + FindReachableTiles() : List<Tile> |

#### PoissonDiscSampling

|  |
| --- |
| **Attributes:** |
| **Methods:**  + GetPoints() : List<Vector2>  - IsValid() : bool  - SquaredDistance() : double |

#### GameData

|  |
| --- |
| **Attributes:**  + Computer : AIData  + GameStateData : GameStateData  + Map : List<TileData>  + MapHeight : int  + MapWidth : int  + Players : List<PlayerData>  + Structures : List<TileData>  + Units : List<UnitData> |
| **Methods:**  + GameData() |

#### GameStateData

|  |
| --- |
| **Attributes:**  + AddAI : bool  + CurrentPlayerIndex : int  + TurnNumber : int |
| **Methods:**  + GameStateData() |

#### PlayerData

|  |
| --- |
| **Attributes:**  + Colour : string  + Funds : int  + HasHQ : bool  + IsAI : bool  + Team |
| **Methods:**  + FromPlayerData() : Player  + PlayerData() |

#### AIData : PlayerData

|  |
| --- |
| **Attributes:**  + AIState : AIState |
| **Methods:**  + AIData()  + FromAIData() : AI |

#### UnitData

|  |
| --- |
| **Attributes:**  + Ammo : int  + CostToProduce : int  + Defence : float  + Fuel : int  + Health : int  + MovementPoints : int  + Position : Vector2  + State : UnitState  + Team : int  + UnitTypeInt : int |
| **Methods:**  + FromUnitData() : Unit  + UnitData() |

#### TileData

|  |
| --- |
| **Attributes:**  + MapGridPos : Vector2  + Position : Vector2 + Team : int  + Type : TileType |
| **Methods:**  + FromTileData() : Tile  + TileData() |

## Data Structures

### Graph

The map can be described as a weighted grid graph. Each cell on the grid represents a vertex in the graph, and its neighbours (up, down, left, and right) are connected to it by edges. Each edge has a weight associated with it (movement cost) which is used to implement the Dijkstra pathfinding algorithm.

### Lists

There are many lists utilised in the game. The primary lists being units in the game; buttons; players, reachable tiles, and attackable tiles.

### Dictionary

Since base damage values vary depending on the attacking and defending unit, a dictionary is an appropriate and efficient way of storing these values.

### Priority Queue

In the implementation Dijkstra's algorithm, a priority queue is used to keep track of the tiles to be explored in order of their total cost from the starting tile, allowing the algorithm to efficiently explore the graph and find the shortest paths from the starting tile to all reachable tiles.

## User Interface

### Main Menu

Graphical user interface, chart

Description automatically generated

The main menu is simple in design with 3 options for the player to choose

### New Game Menu

Diagram

Description automatically generated with medium confidence

In the New Game page, there are multiple options for the player to choose from when creating the game: Refreshing the map design, changing the number of players, increasing/decreasing the map size, and choosing whether to have an AI player or not.

If the player wishes to, they can return to the main menu to either quit or load the previous game that was saved.

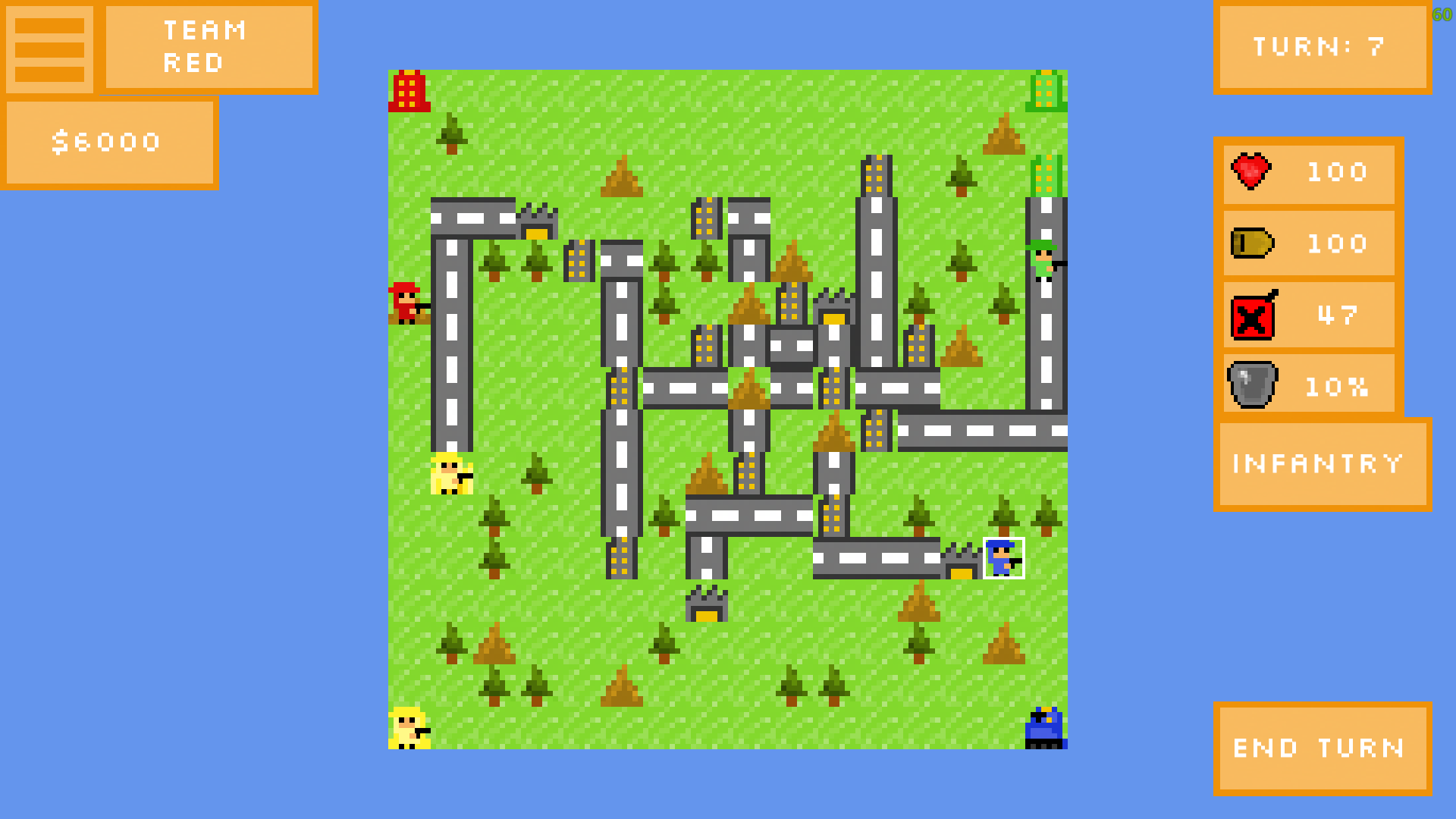
### In Game

A screenshot of a video game

Description automatically generated

When playing the game, all the information that the player requires is displayed: selected unit statistics, current playing team, available funds and turn number.

After a unit has been selected the player can choose what they wish the unit to do from a list of options displayed on the side



Unit statistics other than the current player’s own units can also be displayed which can determine whether the player wishes to risk attacking the unit or not.

A screenshot of a video game

Description automatically generated

When producing a new unit, the list of units that can be produced is displayed and the player may select which unit they want to create, give that they have the required funds to do so.

### Paused Menu

A screenshot of a video game

Description automatically generated

If the player wishes to pause the game, they are presented with the options to either Resume (Continue playing the game), Save (Save the game), Menu (Return to the main menu to either start a new game or load a game) or Quit the application.

## Saving/Loading Games

### Saving Requirements

The game can be split into 5 major sections that require data to be saved for the game to be loaded and played again exactly from where it was left off

Game State

* The current player
* Turn number
* Whether an AI is present

Game Map

* List of all the tiles
* Tile position
* Type
* Grid coordinates in the 2D array
* Team
* Map Width
* Map Height

Units

* List of all the units
* Unit position
* State
* Team
* UnitType as an integer
* Health
* Ammo
* Fuel
* Defence
* MovementPoints
* CostToProduce

Players

* The funds of each player
* The player’s team
* Funds
* Whether the player has an HQ
* Colour
* Whether the player is an AI

AI

* Same as the Player Data
* Current AI State

All this data can then be stored in a singular GameData file.

### Save File Examples

The following are examples of how the data would be structured:

#### Game State

<GameStateData>

<TurnNumber>10</TurnNumber>

<CurrentPlayerIndex>0</CurrentPlayerIndex>

<AddAI>true</AddAI>

</GameStateData>

#### Game Map

*Note that this is the data for a singular Tile. The entire map would consist of many more instances of TileData.*

<Map>

<TileData>

<Type>HQ</Type>

<Position>

<X>512</X>

<Y>92</Y>

</Position>

<MapGridPos>

<X>0</X>

<Y>0</Y>

</MapGridPos>

<Team>0</Team>

</TileData>

<Map>

#### Unit

*Note that this is the data for a singular unit. The list of units would consist of many more instances of UnitData.*

<Units>

<UnitData>

<Position>

<X>792</X>

<Y>316</Y>

</Position>

<State>None</State>

<Team>1</Team>

<UnitTypeInt>1</UnitTypeInt>

<Health>100</Health>

<Ammo>100</Ammo>

<Fuel>42</Fuel>

<Defence>0</Defence>

<MovementPoints>3</MovementPoints>

<CostToProduce>1000</CostToProduce>

</UnitData>

<Units>

#### Players

*Player Data for two players*

<Players>

<PlayerData>

<Team>0</Team>

<Funds>10000</Funds>

<HasHQ>true</HasHQ>

<Colour>Red</Colour>

<IsAI>false</IsAI>

</PlayerData>

<PlayerData>

<Team>1</Team>

<Funds>13000</Funds>

<HasHQ>true</HasHQ>

<Colour>Blue</Colour>

<IsAI>true</IsAI>

</PlayerData>

</Players>

#### AI

<Computer>

<Team>1</Team>

<Funds>5000</Funds>

<HasHQ>true</HasHQ>

<Colour>Blue</Colour>

<IsAI>false</IsAI>

<AIState>Initial</AIState>

</Computer>

### Classes for saving and loading

Serializable classes are required for saving and loading game data in XML format.

Each class has two constructors: One for use by the XmlSerializer and the other for converting the attributes into the data.

|  |  |
| --- | --- |
| GameStateData | Stores the necessary data to restore the previous state of the game |
| TileData | Stores all the necessary data for Tiles |
| UnitData | Stores all the necessary data for Units |
| PlayerData | Stores all the necessary data for Players |
| AIData | Inherits data from players and has its own AIState data value |
| GameData | Contains |

Each class also has a method which returns the same object with all the attributes that were stored as data.

## AI

### AI model

The model I have chosen for the AI is to use a simple state machine as it’s easy to understand and implement. Furthermore, state machines are relatively easy to scale up and extend by adding more states. However, there are various disadvantages to this model such as the lack of flexibility since the AI is unable to adapt dynamically in response to certain situations and the lack of being able to learn different strategies.

### States

The AI has four different states:

* Initial
* Attack
* Defend
* CaptureStructures

#### Initial

The initial state is used at the start of a new game where each player will produce a unit to begin with. This is exactly what the AI will do as well.

|  |
| --- |
| **Explanation:** The AI first gets all the buildings that it owns. It then loops through all its units to attempt to move them away from the HQ, this is to ensure that there is no unit situated on the HQ when attempting to produce a new unit so that at least one unit is always produced per turn.  The algorithm then loops through the list of friendly buildings and generates a unit on each factory and HQ that it owns.  The type of unit that it produces is based on the cost. |
| **Pseudocode:**  SUBROUTINE ProduceUnits()  GetFriendlyBuildings()  FOREACH unit IN AIUnits  IF unit.State != UnitState.Moved AND unit.State != UnitState.Used AND unit.Fuel > 0  MoveToRandomReachable(unit, true)  ENDIF  ENDFOR  FOREACH structure IN FriendlyBuildings  IF structure.Type == TileType.Factory OR structure.Type == TileType.HQ) AND \_inputController.GetTileUnit(structure) == null  unitType = -1    IF Funds >= 7000  Funds -= 7000  unitType = 3  ELSEIF Funds >= 5000  CONTINUE  ELSEIF Funds >= 4000  Funds -= 3000  unitType = 2  ELSE  Funds -= 1000  unitType = 1  ENDIF    IF unitType != -1  newUnit = new(Texture, structure.Position, unitType, Team)  newUnit.State = UnitState.Used  \_unitManager.AddUnit(newUnit)  ENDIF  ENDIF  ENDFOR  ENDSUBROUTINE |

#### Attack

|  |
| --- |
| **Explanation:**  When the required conditions are met, the AI loops through each of its units and attempts to move them towards the enemy HQ to attack (3.6.1 Lines 314 – 343)  However, blocking the unit types of tank and apc of moving specifically to the HQ tile is necessary in order to not block the units that are able to capture the HQ. These units are made to move to a random reachable tile away from the HQ (3.6.1 Lines 173 – 215) |
| **Pseudocode:**  FOREACH Unit unit IN AIUnits  IF unit.State != UnitState.Moved  AND unit.State != UnitState.Used  AND unit.Fuel > 0  IF unit.Type != UnitType.Tank  AND unit.Type != UnitType.APC  MoveTowardsTile(unit, HQToAttack);    ELSE    MoveToRandomReachable(unit, true);  ENDIF  ENDIF  ENDFOR |

#### Defend

|  |
| --- |
| **Explanation:**  When the required conditions are met, the AI loops through each of its units and attempts to move them back towards its own HQ to defend it |
| **Pseudocode:**  FOREACH Unit unit IN AIUnits  IF unit.State != UnitState.Moved  AND unit.State != UnitState.Used  AND unit.Fuel > 0    MoveTowardsTile(unit, HQ);  ENDIF  ENDFOR |

#### CaptureStructures

|  |
| --- |
| **Explanation:**  When the required conditions are met, the AI loops through each of its units and searches for whether there are any structures in reach. If there are, then the AI will move its units towards the structure. If not, then the AI will move its units to a random reachable tile away from the HQ. |
| **Pseudocode:**  FOREACH Unit unit in AIUnits  reachableTiles = GetReachableTiles(unit);  FOREACH Tile structure in closeByStructures  IF reachableTiles.Contains(structure)  AND structure.Team != Team  AND unit.State != UnitState.Moved  AND unit.State != UnitState.Used  AND unit.Type != UnitType.Tank  AND unit.Type != UnitType.APC  AND unit.Fuel > 0  MoveTowardsTile(unit, structure);    ELSEIF unit.State != UnitState.Moved  AND unit.State != UnitState.Used  AND unit.Fuel > 0  MoveToRandomReachable(unit, true);  ENDIF  ENDFOR  ENDFOR |

### Logic Loop

The following is the structure of the AI logic loop

|  |
| --- |
| **Explanation:**  The AI first gets the tiles of the enemy HQs and a list of its own units. It then runs through the logic for its current state executing the algorithms shown prior.  After the state has finished executing, the AI will: attack any enemy units in range of its own units; capture any structures that its units are located at and produce as many units as possible.  It then changes its state according to specified conditions:  If an enemy’s reachable tiles can reach the HQ, then then AI will enter its Defence state. This is the main priority of the AI since if it loses the HQ then it loses the game.  If this is not true, then it can move on to check if it is doing better than the enemy. It does this by going through the list of units and checking whether the AI has more players than the enemy. When this is true then the AI will set the player with less units than the AI as the HQToAttack.  If the AI is neither in an attacking of defending state, then it’ll try to capture buildings around it to gain more funds. |
| **Pseudocode:**  SUBROUTINE RunAILogic()  GetEnemyHQs()  GetAIUnits()  SWITCH State:  CASE AIState.Initial:  (ProduceUnits)  BREAK  CASE AIState.Attack:  (Attack)  BREAK  CASE AIState.Defend:  (Defend)  BREAK  CASE AIState.CaptureStructures:  (CaptureStructures)  BREAK  ENDSWITCH  AttackNeighbouringUnits()  CaptureStructures()  ProduceUnits()  IF EnemyCloseToHQ()  State = AIState.Defend  ELSE IF Winning()  State = AIState.Attack  ELSE IF CloseByStructure()  State = AIState.CaptureStructures  ENDIF  ENDSUBROUTINE |

# Technical Solution

## Areas of interest

The following table lists areas of interest within the code. Note that this is not an exhaustive list and there may be other areas of interest

|  |  |  |
| --- | --- | --- |
| Name | File | Line number |
| Main game loop | BasicWarsGame.cs | 110 – 190, 203 – 316 |
| Save Game | BasicWarsGame.cs | 705 – 762 |
| Load Game | BasicWarsGame.cs | 764 – 843 |
| Player Select | BasicWarsGame.cs | 331 – 393 |
| Player Attack | BasicWarsGame.cs | 485 – 529 |
| Damage Dictionary | GameUI.cs | 11 – 25 |
| Player Move | BasicWarsGame.cs | 447 – 483 |
| Player Resupply | BasicWarsGame.cs | 549 – 584 |
| Get reachable tiles | GameUI.cs | 260 – 315 |
| Get attackable tiles | GameUI.cs | 317 – 342 |
| AI game logic | AI.cs | 49 – 150 |
| Get points | PoissonDiscSampling.cs | 9 – 58 |
| FindReachableTiles | Dijkstra.cs | 15 – 43 |
| Generate roads | MapManager.cs | 166 – 178 |
| Generate structures | Mapmanager.cs | 127 – 164 |

## Code

### BasicWarsGame.cs

1. **using** Basic\_Wars\_V2**.**Entities**;**
2. **using** Basic\_Wars\_V2**.**Enums**;**
3. **using** Basic\_Wars\_V2**.**System**;**
4. **using** Microsoft**.**Xna**.**Framework**;**
5. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
6. **using** System**.**Collections**.**Generic**;**
7. **using** System**.**IO**;**
8. **using** System**.**Xml**.**Serialization**;**
9. **namespace** Basic\_Wars\_V2
10. **{**
11. **public** class BasicWarsGame **:** Game
12. **{**
13. **private** **readonly** GraphicsDeviceManager \_graphics**;**
14. **private** SpriteBatch \_spriteBatch**;**
15. **private** const string ASSET\_NAME\_IN\_GAME\_ASSETS **=** "InGameAssets"**;**
16. **private** const string ASSET\_NAME\_GAMEFONT **=** "Font"**;**
17. **private** const string SAVE\_GAME\_PATH **=** "GameData.xml"**;**
18. **private** Texture2D InGameAssets**;**
19. **private** SpriteFont Font**;**
20. **private** const int WINDOW\_WIDTH **=** 1920**;**
21. **private** const int WINDOW\_HEIGHT **=** 1080**;**
22. **private** GameState gameState**;**
23. **private** MenuState menuState**;**
24. **private** List**<**Player**>** Players**;**
25. **private** AI Computer**;**
26. **private** bool AddAI**;**
27. **private** Player CurrentPlayer**;**
28. **private** int CurrentPlayerIndex**;**
29. **private** bool NextPlayer**;**
30. **private** InputController \_inputController**;**
31. **private** bool ProcessButtonsOnly**;**
32. **private** GameUI \_gameUI**;**
33. **private** EntityManager \_entityManager**;**
34. **private** UnitManager \_unitManager**;**
35. **private** ButtonManager \_buttonManager**;**
36. **private** MapManager \_gameMap**;**
37. **private** Unit SelectedUnit**;**
38. **private** Tile CurrentUnitTile**;**
39. **private** List**<**Tile**>** reachableTiles**;**
40. **private** List**<**Tile**>** attackableTiles**;**
41. **private** Tile SelectedTile**;**
42. **private** Button PressedButton**;**
43. **private** int TurnNumber**;**
44. **private** bool DrawRan**;**
45. **public** BasicWarsGame**()**
46. **{**
47. \_graphics **=** **new** GraphicsDeviceManager**(this);**
48. Content**.**RootDirectory **=** "Content"**;**
49. IsMouseVisible **=** **true;**
50. **}**
51. **protected** **override** void Initialize**()**
52. **{**
53. \_graphics**.**IsFullScreen **=** **false;**
54. IsFixedTimeStep **=** **true;**
55. \_graphics**.**PreferredBackBufferWidth **=** WINDOW\_WIDTH**;**
56. \_graphics**.**PreferredBackBufferHeight **=** WINDOW\_HEIGHT**;**
57. \_graphics**.**ApplyChanges**();**
58. **base.**Initialize**();**
59. **}**
60. **protected** **override** void LoadContent**()**
61. **{**
62. \_spriteBatch **=** **new** SpriteBatch**(**GraphicsDevice**);**
63. InGameAssets **=** Content**.**Load**<**Texture2D**>(**ASSET\_NAME\_IN\_GAME\_ASSETS**);**
64. Font **=** Content**.**Load**<**SpriteFont**>(**ASSET\_NAME\_GAMEFONT**);**
65. menuState **=** MenuState**.**Initial**;**
66. Players **=** **new** List**<**Player**>();**
67. NextPlayer **=** **true;**
68. AddAI **=** **false;**
69. TurnNumber **=** 0**;**
70. \_entityManager **=** **new** EntityManager**();**
71. \_unitManager **=** **new** UnitManager**();**
72. \_buttonManager **=** **new** ButtonManager**();**
73. \_gameMap **=** **new** MapManager**(**InGameAssets**,** 16**,** 16**,** 2**);**
74. \_gameUI **=** **new** GameUI**(**InGameAssets**,** Font**,** \_gameMap**,** \_unitManager**,** \_buttonManager**);**
75. \_inputController **=** **new** InputController**(**\_unitManager**,** \_buttonManager**,** \_gameMap**);**
76. ProcessButtonsOnly **=** **true;**
77. \_entityManager**.**AddEntity**(**\_gameMap**);**
78. \_entityManager**.**AddEntity**(**\_unitManager**);**
79. \_entityManager**.**AddEntity**(**\_gameUI**);**
80. **}**
81. **protected** **override** void Update**(**GameTime gameTime**)**
82. **{**
83. \_inputController**.**ProcessControls**(**gameTime**,** ProcessButtonsOnly**);**
84. PressedButton **=** \_inputController**.**GetButtonPressed**();**
85. \_entityManager**.**Update**(**gameTime**);**
86. **switch** **(**menuState**)**
87. **{**
88. **case** MenuState**.**Initial**:**
89. \_unitManager**.**ClearUnits**();**
90. \_gameMap**.**DrawMap **=** **false;**
91. menuState **=** \_gameUI**.**Init**(**gameTime**,** PressedButton**);**
92. **break;**
93. **case** MenuState**.**NewGame**:**
94. \_unitManager**.**ClearUnits**();**
95. TurnNumber **=** 0**;**
96. \_gameMap**.**DrawMap **=** **true;**
97. menuState **=** \_gameUI**.**NewGame**(**gameTime**,** PressedButton**);**
98. **break;**
99. **case** MenuState**.**AddAI**:**
100. AddAI **=** **true;**
101. menuState **=** MenuState**.**NewGame**;**
102. **break;**
103. **case** MenuState**.**RemoveAI**:**
104. AddAI **=** **false;**
105. menuState **=** MenuState**.**NewGame**;**
106. **break;**
107. **case** MenuState**.**IncreaseMapSize**:**
108. RefreshMap**(**\_gameMap**.**MapWidth **+=** 1**,** \_gameMap**.**MapHeight **+=** 1**);**
109. menuState **=** MenuState**.**NewGame**;**
110. **break;**
111. **case** MenuState**.**DecreaseMapSize**:**
112. **if** **(**\_gameMap**.**MapWidth **>** 6 **&&** \_gameMap**.**MapHeight **>** 6**)**
113. **{**
114. RefreshMap**(**\_gameMap**.**MapWidth **-=** 1**,** \_gameMap**.**MapHeight **-=** 1**);**
115. **}**
116. menuState **=** MenuState**.**NewGame**;**
117. **break;**
118. **case** MenuState**.**SaveGame**:**
119. SaveGame**(**gameTime**);**
120. **break;**
121. **case** MenuState**.**PlayingGame**:**
122. PlayingGame**(**gameTime**);**
123. **break;**
124. **case** MenuState**.**GamePaused**:**
125. ProcessButtonsOnly **=** **true;**
126. menuState **=** \_gameUI**.**PausedGame**(**gameTime**,** PressedButton**);**
127. gameState **=** GameState**.**PlayerSelect**;**
128. **break;**
129. **case** MenuState**.**GameOver**:**
130. ProcessButtonsOnly **=** **true;**
131. menuState **=** \_gameUI**.**GameOver**(**gameTime**,** PressedButton**,** CurrentPlayer**);**
132. **break;**
133. **case** MenuState**.**RefreshMap**:**
134. RefreshMap**(**\_gameMap**.**MapWidth**,** \_gameMap**.**MapHeight**);**
135. **break;**
136. **case** MenuState**.**LoadGame**:**
137. LoadGame**(**gameTime**);**
138. **break;**
139. **case** MenuState**.**QuitGame**:**
140. Exit**();**
141. **break;**
142. **}**
143. DrawRan **=** **true;**
144. **base.**Update**(**gameTime**);**
145. **}**
146. **protected** **override** void Draw**(**GameTime gameTime**)**
147. **{**
148. GraphicsDevice**.**Clear**(**Color**.**CornflowerBlue**);**
149. \_spriteBatch**.**Begin**();**
150. \_entityManager**.**Draw**(**\_spriteBatch**,** gameTime**);**
151. \_spriteBatch**.**End**();**
152. **base.**Draw**(**gameTime**);**
153. **}**
154. **private** void PlayingGame**(**GameTime gameTime**)**
155. **{**
156. **if** **(**TurnNumber **==** 0**)**
157. **{**
158. TurnNumber **=** 1**;**
159. Players **=** \_gameUI**.**GetPlayers**();**
160. \_unitManager**.**DrawUnits **=** **true;**
161. CurrentPlayerIndex **=** 0**;**
162. NextPlayer **=** **true;**
163. **if** **(**AddAI**)**
164. **{**
165. Players**[**Players**.**Count **-** 1**].**IsAI **=** **true;**
166. Computer **=** **new** AI**(**Players**.**Count **-** 1**,** 0**,** \_gameMap**,** \_unitManager**,** \_gameUI**,** \_inputController**,** InGameAssets**);**
167. **}**
168. **}**
169. **if** **(**NextPlayer**)**
170. **{**
171. \_unitManager**.**ResetUnitStates**(**CurrentPlayer**);**
172. \_gameUI**.**DrawSelectedUI **=** **false;**
173. NextPlayer **=** **false;**
174. **if** **(**CurrentPlayerIndex **>** Players**.**Count **-** 1**)**
175. **{**
176. TurnNumber**++;**
177. **}**
178. **if** **(**CurrentPlayerIndex **>** Players**.**Count **-** 1**)**
179. **{**
180. CurrentPlayerIndex **=** 0**;**
181. **}**
182. CurrentPlayer **=** Players**[**CurrentPlayerIndex**];**
183. Income**(**CurrentPlayer**);**
184. **if** **(**CurrentPlayer**.**IsAI**)**
185. **{**
186. gameState **=** GameState**.**AITurn**;**
187. **}**
188. **else**
189. **{**
190. gameState **=** \_gameUI**.**Turn**(**gameTime**,** CurrentPlayer**,** TurnNumber**,** PressedButton**);**
191. **}**
192. **}**
193. **switch** **(**gameState**)**
194. **{**
195. **case** GameState**.**PlayerSelect**:**
196. gameState **=** \_gameUI**.**Turn**(**gameTime**,** CurrentPlayer**,** TurnNumber**,** PressedButton**);**
197. PlayerSelect**(**gameTime**);**
198. **break;**
199. **case** GameState**.**SelectAction**:**
200. PlayerSelectAction**(**gameTime**,** PressedButton**);**
201. **break;**
202. **case** GameState**.**UnitIdle**:**
203. SelectedUnit**.**State **=** UnitState**.**Used**;**
204. gameState **=** GameState**.**PlayerSelect**;**
205. **break;**
206. **case** GameState**.**PlayerMove**:**
207. PlayerMove**(**gameTime**);**
208. **break;**
209. **case** GameState**.**PlayerAttack**:**
210. PlayerAttack**(**gameTime**);**
211. **break;**
212. **case** GameState**.**PlayerCapture**:**
213. PlayerCapture**(**gameTime**);**
214. **break;**
215. **case** GameState**.**PlayerResupply**:**
216. PlayerResupply**(**gameTime**);**
217. **break;**
218. **case** GameState**.**PlayerProduceUnit**:**
219. PlayerProduceUnit**(**gameTime**);**
220. **break;**
221. **case** GameState**.**PauseGame**:**
222. menuState **=** MenuState**.**GamePaused**;**
223. \_gameUI**.**DrawSelectedUI **=** **false;**
224. **break;**
225. **case** GameState**.**EnemyTurn**:**
226. NextPlayer **=** **true;**
227. CurrentPlayerIndex**++;**
228. **break;**
229. **case** GameState**.**AITurn**:**
230. **foreach** **(**Tile structure **in** \_gameMap**.**Structures**)**
231. **{**
232. **if** **(**structure**.**Team **==** Computer**.**Team**)**
233. **{**
234. Computer**.**Funds **+=** 1000**;**
235. **}**
236. **}**
237. Computer**.**RunAILogic**();**
238. bool GameOver **=** CheckWinner**(**Computer**.**Team**);**
239. **if** **(**GameOver**)**
240. **{**
241. menuState **=** MenuState**.**GameOver**;**
242. **}**
243. **else**
244. **{**
245. gameState **=** GameState**.**EnemyTurn**;**
246. **}**
247. **break;**
248. **}**
249. **}**
250. **private** void RefreshMap**(**int Width **=** 16**,** int Height **=** 16**)**
251. **{**
252. Players **=** \_gameUI**.**GetPlayers**();**
253. \_entityManager**.**RemoveEntity**(**\_gameMap**);**
254. \_gameMap **=** **new** MapManager**(**InGameAssets**,** Width**,** Height**,** Players**.**Count**);**
255. \_entityManager**.**AddEntity**(**\_gameMap**);**
256. \_gameUI**.**ChangeMap**(**\_gameMap**);**
257. \_inputController**.**ChangeMap**(**\_gameMap**);**
258. menuState **=** MenuState**.**NewGame**;**
259. **}**
260. **private** void PlayerSelect**(**GameTime gameTime**)**
261. **{**
262. ProcessButtonsOnly **=** **false;**
263. UpdateUnitStats**();**
264. \_gameUI**.**ClearAttackableOverlay**();**
265. //For if return or idle is selected as a unit action
266. **if** **(**SelectedUnit **!=** **null)**
267. **{**
268. \_gameUI**.**DisplayAttributes**(**SelectedUnit**);**
269. **if** **(**SelectedUnit**.**State **!=** UnitState**.**Used
270. **&&** SelectedUnit**.**State **!=** UnitState**.**Moved
271. **&&** SelectedUnit**.**State **!=** UnitState**.**Dead
272. **)**
273. **{**
274. SelectedUnit**.**State **=** UnitState**.**None**;**
275. **}**
276. SelectedUnit**.**Selected **=** **false;**
277. **}**
278. SelectedUnit **=** \_inputController**.**GetSelectedUnit**();**
279. SelectedTile **=** \_inputController**.**GetSelectedTile**();**
280. **if** **(**SelectedUnit **!=** **null)**
281. **{**
282. \_gameUI**.**ChangeSelectedPosition**(**SelectedUnit**.**Position**);**
283. \_gameUI**.**DrawSelectedUI **=** **true;**
284. SelectedUnit**.**Selected **=** **true;**
285. \_gameUI**.**DisplayAttributes**(**SelectedUnit**);**
286. **if** **(**SelectedUnit**.**Team **==** CurrentPlayer**.**Team
287. **&&** SelectedUnit**.**State **!=** UnitState**.**Used
288. **)**
289. **{**
290. gameState **=** GameState**.**SelectAction**;**
291. **}**
292. **else**
293. **{**
294. SelectedUnit**.**Selected **=** **false;**
295. **}**
296. **}**
297. **if** **(**SelectedTile **!=** **null)**
298. **{**
299. \_gameUI**.**ChangeSelectedPosition**(**SelectedTile**.**Position**);**
300. \_gameUI**.**DrawSelectedUI **=** **true;**
301. \_gameUI**.**DisplayAttributes**(null,** SelectedTile**);**
302. **if** **((**SelectedTile**.**Type **==** TileType**.**Factory
303. **||** SelectedTile**.**Type **==** TileType**.**HQ**)**
304. **&&** SelectedTile**.**Team **==** CurrentPlayer**.**Team
305. **)**
306. **{**
307. ProcessButtonsOnly **=** **true;**
308. gameState **=** GameState**.**PlayerProduceUnit**;**
309. **}**
310. **}**
311. **}**
312. **private** void PlayerSelectAction**(**GameTime gameTime**,** Button PressedButton**)**
313. **{**
314. CurrentUnitTile **=** \_inputController**.**GetUnitTile**(**SelectedUnit**);**
315. ProcessButtonsOnly **=** **true;**
316. \_gameUI**.**DisplayAttributes**(**SelectedUnit**);**
317. **if** **(**SelectedUnit**.**State **!=** UnitState**.**Moved
318. **&&** SelectedUnit**.**State **!=** UnitState**.**Used
319. **)**
320. **{**
321. reachableTiles **=** \_gameUI**.**GetReachableTiles**(**SelectedUnit**,** \_unitManager**.**GetUnitPositions**(),** CurrentUnitTile**);**
322. **}**
323. **else**
324. **{**
325. reachableTiles**.**Clear**();**
326. \_gameUI**.**ClearMoveableOverlay**();**
327. **}**
328. attackableTiles **=** \_gameUI**.**GetAttackableTiles**(**SelectedUnit**,** CurrentUnitTile**);**
329. bool displayCapture **=** **false;**
330. bool displayResupply **=** **false;**
331. **if** **((**CurrentUnitTile**.**Type **==** TileType**.**City
332. **||** CurrentUnitTile**.**Type **==** TileType**.**Factory
333. **||** CurrentUnitTile**.**Type **==** TileType**.**HQ**)**
334. **&&** SelectedUnit**.**Type **!=** UnitType**.**Tank
335. **&&** SelectedUnit**.**Type **!=** UnitType**.**APC
336. **&&** SelectedUnit**.**Team **!=** CurrentUnitTile**.**Team
337. **)**
338. **{**
339. displayCapture **=** **true;**
340. **}**
341. **if** **(**
342. **(**
343. **(**CurrentUnitTile**.**Type **==** TileType**.**City
344. **||** CurrentUnitTile**.**Type **==** TileType**.**Factory
345. **||** CurrentUnitTile**.**Type **==** TileType**.**HQ
346. **)**
347. **&&** SelectedUnit**.**Team **==** CurrentUnitTile**.**Team
348. **)**
349. **||** SelectedUnit**.**Type **==** UnitType**.**APC
350. **)**
351. **{**
352. displayResupply **=** **true;**
353. **}**
354. gameState **=** \_gameUI**.**DisplayPlayerActions**(**gameTime**,** PressedButton**,** displayCapture**,** displayResupply**);**
355. **}**
356. **private** void PlayerMove**(**GameTime gameTime**)**
357. **{**
358. **if** **(**reachableTiles**.**Count **!=** 0**)**
359. **{**
360. **while** **(**gameState **==** GameState**.**PlayerMove **&&** DrawRan**)**
361. **{**
362. DrawRan **=** **false;**
363. **foreach** **(**Tile tile **in** reachableTiles**)**
364. **{**
365. **if** **(**
366. \_inputController**.**MouseCollider**.**Intersects**(**tile**.**Collider**)**
367. **&&** \_inputController**.**LeftMouseClicked**()**
368. **&&** SelectedUnit**.**Fuel **>** 0
369. **)**
370. **{**
371. SelectedUnit**.**Position **=** tile**.**Position**;**
372. SelectedUnit**.**Fuel**--;**
373. SelectedUnit**.**State **=** UnitState**.**Moved**;**
374. \_gameUI**.**ChangeSelectedPosition**(**SelectedUnit**.**Position**);**
375. gameState **=** GameState**.**SelectAction**;**
376. **}**
377. **}**
378. **if** **(**SelectedUnit**.**Fuel **<=** 0**)**
379. **{**
380. gameState **=** GameState**.**SelectAction**;**
381. **}**
382. **}**
383. **}**
384. **else**
385. **{**
386. gameState **=** GameState**.**PlayerSelect**;**
387. **}**
388. **}**
389. **private** void PlayerAttack**(**GameTime gameTime**)**
390. **{**
391. **if** **(**SelectedUnit**.**State **!=** UnitState**.**Used
392. **&&** SelectedUnit**.**Type **!=** UnitType**.**APC
393. **&&** attackableTiles**.**Count **!=** 0
394. **)**
395. **{**
396. **while** **(**gameState **==** GameState**.**PlayerAttack **&&** DrawRan**)**
397. **{**
398. DrawRan **=** **false;**
399. **foreach** **(**Tile tile **in** attackableTiles**)**
400. **{**
401. **if** **(**
402. \_inputController**.**MouseCollider**.**Intersects**(**tile**.**Collider**)**
403. **&&** \_inputController**.**LeftMouseClicked**()**
404. **&&** SelectedUnit**.**Ammo **>** 0
405. **)**
406. **{**
407. Unit defendingUnit **=** \_inputController**.**GetTileUnit**(**tile**);**
408. defendingUnit**.**Health **-=** \_gameUI**.**CalculateDamage**(**SelectedUnit**,** defendingUnit**);**
409. **if** **(**defendingUnit**.**Health **>** 0 **&&** defendingUnit**.**Ammo **>** 0**)**
410. **{**
411. SelectedUnit**.**Health **-=** \_gameUI**.**CalculateDamage**(**defendingUnit**,** SelectedUnit**);**
412. defendingUnit**.**Ammo**--;**
413. **}**
414. SelectedUnit**.**State **=** UnitState**.**Used**;**
415. gameState **=** GameState**.**PlayerSelect**;**
416. **}**
417. **if** **(**SelectedUnit**.**Ammo **<=** 0**)**
418. **{**
419. gameState **=** GameState**.**SelectAction**;**
420. **}**
421. **}**
422. **}**
423. **}**
424. **else**
425. **{**
426. gameState **=** GameState**.**PlayerSelect**;**
427. **}**
428. **}**
429. **private** void PlayerCapture**(**GameTime gameTime**)**
430. **{**
431. CurrentUnitTile**.**Team **=** SelectedUnit**.**Team**;**
432. SelectedUnit**.**State **=** UnitState**.**Used**;**
433. CurrentUnitTile**.**CreateTileSpriteOnType**();**
434. bool GameOver **=** CheckWinner**(**CurrentPlayer**.**Team**);**
435. CheckHQ**();**
436. **if** **(**GameOver**)**
437. **{**
438. menuState **=** MenuState**.**GameOver**;**
439. **}**
440. gameState **=** GameState**.**PlayerSelect**;**
441. **}**
442. **private** void PlayerResupply**(**GameTime gameTime**)**
443. **{**
444. **if** **(**SelectedUnit**.**Type **==** UnitType**.**APC**)**
445. **{**
446. SelectedUnit**.**State **=** UnitState**.**Used**;**
447. List**<**Tile**>** neighbours **=** \_gameMap**.**GetNeighbours**(**CurrentUnitTile**);**
448. **foreach** **(**Tile neighbour **in** neighbours**)**
449. **{**
450. Unit neighbouringUnit **=** \_inputController**.**GetTileUnit**(**neighbour**);**
451. **if** **(**neighbouringUnit **!=** **null** **&&** neighbouringUnit**.**Team **==** SelectedUnit**.**Team**)**
452. **{**
453. neighbouringUnit**.**RefreshUnitAttributes**();**
454. **}**
455. **}**
456. **if** **((**CurrentUnitTile**.**Type **==** TileType**.**City
457. **||** CurrentUnitTile**.**Type **==** TileType**.**Factory
458. **||** CurrentUnitTile**.**Type **==** TileType**.**HQ**)**
459. **&&** CurrentUnitTile**.**Team **==** SelectedUnit**.**Team
460. **)**
461. **{**
462. SelectedUnit**.**RefreshUnitAttributes**();**
463. **}**
464. gameState **=** GameState**.**PlayerSelect**;**
465. **}**
466. **else**
467. **{**
468. SelectedUnit**.**State **=** UnitState**.**Used**;**
469. SelectedUnit**.**RefreshUnitAttributes**();**
470. gameState **=** GameState**.**PlayerSelect**;**
471. **}**
472. **}**
473. **private** void PlayerProduceUnit**(**GameTime gameTime**)**
474. **{**
475. **while** **(**gameState **==** GameState**.**PlayerProduceUnit **&&** DrawRan**)**
476. **{**
477. DrawRan **=** **false;**
478. int unitType **=** \_gameUI**.**ProcessUnitProduction**(**gameTime**,** PressedButton**);**
479. **if** **(**unitType **==** **-**1**)**
480. **{**
481. gameState **=** GameState**.**PlayerSelect**;**
482. **}**
483. **if** **(**unitType **!=** **-**1 **&&** unitType **!=** **-**2**)**
484. **{**
485. Unit newUnit **=** **new(**InGameAssets**,** SelectedTile**.**Position**,** unitType**,** CurrentPlayer**.**Team**);**
486. **if** **(**CurrentPlayer**.**Funds **>=** newUnit**.**CostToProduce**)**
487. **{**
488. CurrentPlayer**.**Funds **-=** newUnit**.**CostToProduce**;**
489. newUnit**.**State **=** UnitState**.**Used**;**
490. \_unitManager**.**AddUnit**(**newUnit**);**
491. gameState **=** GameState**.**PlayerSelect**;**
492. **}**
493. **}**
494. **}**
495. **}**
496. **private** void UpdateUnitStats**()**
497. **{**
498. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
499. **{**
500. unit**.**Defence **=** \_inputController**.**GetUnitTile**(**unit**).**DefenceBonus**;**
501. **}**
502. **}**
503. **private** void Income**(**Player player**)**
504. **{**
505. **foreach** **(**Tile structure **in** \_gameMap**.**Structures**)**
506. **{**
507. **if** **(**structure**.**Team **==** player**.**Team**)**
508. **{**
509. player**.**Funds **+=** 1000**;**
510. **}**
511. **}**
512. **}**
513. **private** bool CheckWinner**(**int Team**)**
514. **{**
515. **foreach** **(**Tile HQ **in** \_gameMap**.**HQs**)**
516. **{**
517. **if** **(**HQ**.**Team **!=** Team**)**
518. **{**
519. **return** **false;**
520. **}**
521. **}**
522. **return** **true;**
523. **}**
524. **private** void CheckHQ**()**
525. **{**
526. List**<**Player**>** playersToRemove **=** **new();**
527. **foreach** **(**Player player **in** Players**)**
528. **{**
529. **foreach** **(**Tile HQ **in** \_gameMap**.**HQs**)**
530. **{**
531. **if** **(**HQ**.**Team **==** player**.**Team**)**
532. **{**
533. player**.**HasHQ **=** **true;**
534. **break;**
535. **}**
536. **else**
537. **{**
538. player**.**HasHQ **=** **false;**
539. **}**
540. **}**
541. **if** **(!**player**.**HasHQ**)**
542. **{**
543. playersToRemove**.**Add**(**player**);**
544. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
545. **{**
546. **if** **(**unit**.**Team **==** player**.**Team**)**
547. **{**
548. \_unitManager**.**RemoveUnit**(**unit**);**
549. **}**
550. **}**
551. **foreach** **(**Tile structure **in** \_gameMap**.**Structures**)**
552. **{**
553. **if** **(**structure**.**Team **==** player**.**Team**)**
554. **{**
555. structure**.**Team **=** **-**1**;**
556. **switch** **(**structure**.**Type**)**
557. **{**
558. **case** TileType**.**City**:**
559. structure**.**CreateTileSprite**(-**6**);**
560. **break;**
561. **case** TileType**.**Factory**:**
562. structure**.**CreateTileSprite**(-**6**,** 1**);**
563. **break;**
564. **}**
565. **}**
566. **}**
567. **}**
568. **}**
569. **foreach** **(**Player player **in** playersToRemove**)**
570. **{**
571. Players**.**Remove**(**player**);**
572. **}**
573. **}**
574. **private** void SaveGame**(**GameTime gameTime**)**
575. **{**
576. List**<**UnitData**>** unitData **=** **new();**
577. List**<**TileData**>** mapData **=** **new();**
578. List**<**TileData**>** structuresData **=** **new();**
579. List**<**PlayerData**>** playerData **=** **new();**
580. GameStateData gameStateData **=** **new(**TurnNumber**,** CurrentPlayerIndex**,** AddAI**);**
581. AIData ComputerData **=** **null;**
582. **if** **(**AddAI**)**
583. **{**
584. ComputerData **=** **new** AIData**(**Computer**);**
585. **}**
586. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
587. **{**
588. unitData**.**Add**(new** UnitData**(**unit**));**
589. **}**
590. **foreach** **(**Tile tile **in** \_gameMap**.**Map**)**
591. **{**
592. mapData**.**Add**(new** TileData**(**tile**));**
593. **}**
594. **foreach** **(**Tile tile **in** \_gameMap**.**Structures**)**
595. **{**
596. **if** **(**tile**.**Type **!=** TileType**.**HQ**)**
597. **{**
598. structuresData**.**Add**(new** TileData**(**tile**));**
599. **}**
600. **}**
601. **foreach** **(**Player player **in** Players**)**
602. **{**
603. playerData**.**Add**(new** PlayerData**(**player**));**
604. **}**
605. GameData gameData**;**
606. **if** **(**AddAI**)**
607. **{**
608. gameData **=** **new** GameData**(**unitData**,** mapData**,** structuresData**,** playerData**,** gameStateData**,** \_gameMap**.**MapWidth**,** \_gameMap**.**MapHeight**,** ComputerData**);**
609. **}**
610. **else**
611. **{**
612. gameData **=** **new** GameData**(**unitData**,** mapData**,** structuresData**,** playerData**,** gameStateData**,** \_gameMap**.**MapWidth**,** \_gameMap**.**MapHeight**);**
613. **}**
614. XmlSerializer serializer **=** **new(typeof(**GameData**));**
615. **using** **(**StreamWriter streamWriter **=** **new(**SAVE\_GAME\_PATH**))**
616. **{**
617. serializer**.**Serialize**(**streamWriter**,** gameData**);**
618. **}**
619. menuState **=** MenuState**.**PlayingGame**;**
620. **}**
621. **private** void LoadGame**(**GameTime gameTime**)**
622. **{**
623. **if** **(**File**.**Exists**(**SAVE\_GAME\_PATH**))**
624. **{**
625. \_unitManager**.**DrawUnits **=** **true;**
626. \_gameMap**.**DrawMap **=** **true;**
627. \_unitManager**.**ClearUnits**();**
628. Players**.**Clear**();**
629. \_gameMap**.**ClearMap**();**
630. \_gameMap**.**Structures**.**Clear**();**
631. \_gameMap**.**HQs**.**Clear**();**
632. GameData gameData**;**
633. List**<**Tile**>** Map **=** **new();**
634. XmlSerializer serializer **=** **new(typeof(**GameData**));**
635. **using** **(**StreamReader streamReader **=** **new(**SAVE\_GAME\_PATH**))**
636. **{**
637. gameData **=** **(**GameData**)**serializer**.**Deserialize**(**streamReader**);**
638. **}**
639. **foreach** **(**UnitData unitData **in** gameData**.**Units**)**
640. **{**
641. \_unitManager**.**AddUnit**(**unitData**.**FromUnitData**(**InGameAssets**));**
642. **}**
643. **foreach** **(**TileData tileData **in** gameData**.**Map**)**
644. **{**
645. Map**.**Add**(**tileData**.**FromTileData**(**InGameAssets**));**
646. **}**
647. **foreach** **(**PlayerData players **in** gameData**.**Players**)**
648. **{**
649. Players**.**Add**(**players**.**FromPlayerData**());**
650. **}**
651. **foreach** **(**TileData structure **in** gameData**.**Structures**)**
652. **{**
653. \_gameMap**.**Structures**.**Add**(**structure**.**FromTileData**(**InGameAssets**));**
654. **}**
655. \_gameMap**.**MapWidth **=** gameData**.**MapWidth**;**
656. \_gameMap**.**MapHeight **=** gameData**.**MapHeight**;**
657. \_gameMap**.**ResetMapSize**();**
658. **foreach** **(**Tile tile **in** Map**)**
659. **{**
660. \_gameMap**.**Map**[(**int**)**tile**.**MapGridPos**.**X**,** **(**int**)**tile**.**MapGridPos**.**Y**]** **=** tile**;**
661. **if** **(**tile**.**Type **==** TileType**.**HQ**)**
662. **{**
663. \_gameMap**.**HQs**.**Add**(**tile**);**
664. **}**
665. **}**
666. \_gameMap**.**RegenerateMap**();**
667. AddAI **=** gameData**.**GameStateData**.**AddAI**;**
668. CurrentPlayerIndex **=** gameData**.**GameStateData**.**CurrentPlayerIndex**;**
669. TurnNumber **=** gameData**.**GameStateData**.**TurnNumber**;**
670. CurrentPlayer **=** Players**[**CurrentPlayerIndex**];**
671. **if** **(**CurrentPlayer**.**IsAI**)**
672. **{**
673. Computer **=** gameData**.**Computer**.**FromAIData**(**InGameAssets**,** \_gameMap**,** \_unitManager**,** \_gameUI**,** \_inputController**);**
674. gameState **=** GameState**.**AITurn**;**
675. **}**
676. menuState **=** MenuState**.**PlayingGame**;**
677. **}**
678. **else**
679. **{**
680. menuState **=** MenuState**.**Initial**;**
681. **}**
682. **}**
683. **}**
684. **}**

### Graphics

#### Sprite.cs

1. **using** Microsoft**.**Xna**.**Framework**;**
2. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
3. **namespace** Basic\_Wars**.**Graphics
4. **{**
5. **public** class Sprite
6. **{**
7. **public** Texture2D Texture **{** **get;** **set;** **}**
8. **public** int X **{** **get;** **set;** **}**
9. **public** int Y **{** **get;** **set;** **}**
10. **public** int Width **{** **get;** **set;** **}**
11. **public** int Height **{** **get;** **set;** **}**
12. **public** Color TintColour **{** **get;** **set;** **}**
13. **public** Sprite**(**Texture2D texture**,** int x**,** int y**,** int width**,** int height**)**
14. **{**
15. Texture **=** texture**;**
16. X **=** x**;**
17. Y **=** y**;**
18. Width **=** width**;**
19. Height **=** height**;**
20. TintColour **=** Color**.**White**;**
21. **}**
22. **public** void Draw**(**SpriteBatch spriteBatch**,** Vector2 Position**)**
23. **{**
24. spriteBatch**.**Draw**(**Texture**,** Position**,** **new** Rectangle**(**X**,** Y**,** Width**,** Height**),** TintColour**);**
25. **}**
26. **}**
27. **}**

#### Font.cs

1. **using** Microsoft**.**Xna**.**Framework**;**
2. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
3. **namespace** Basic\_Wars\_V2**.**Graphics
4. **{**
5. **public** class Font
6. **{**
7. **public** SpriteFont font **{** **get;** **set;** **}**
8. **public** string Text **{** **get;** **set;** **}**
9. **public** Vector2 TextMiddlePoint **{** **get;** **set;** **}**
10. **public** Color TintColour **{** **get;** **set;** **}** **=** Color**.**White**;**
11. **public** Font**(**SpriteFont font**,** string text**)**
12. **{**
13. **this.**font **=** font**;**
14. Text **=** text**;**
15. TextMiddlePoint **=** font**.**MeasureString**(**text**)** **/** 2**;**
16. **}**
17. **public** void WriteText**(**SpriteBatch \_spriteBatch**,** Vector2 Position**,** float Scale **=** 1.0f**)**
18. **{**
19. \_spriteBatch**.**DrawString**(**font**,** Text**,** Position**,** TintColour**,** 0**,** TextMiddlePoint**,** Scale**,** SpriteEffects**.**None**,** 0.5f**);**
20. **}**
21. **}**
22. **}**

### Entities

#### EntityManager.cs

1. **using** Microsoft**.**Xna**.**Framework**;**
2. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
3. **using** System**.**Collections**.**Generic**;**
4. **using** System**.**Linq**;**
5. **namespace** Basic\_Wars\_V2**.**Entities
6. **{**
7. **internal** class EntityManager
8. **{**
9. **private** List**<**IGameEntity**>** entities **=** **new();**
10. **private** **readonly** List**<**IGameEntity**>** entitiesToRemove **=** **new();**
11. **public** void Update**(**GameTime gameTime**)**
12. **{**
13. **foreach** **(**IGameEntity entity **in** entities**)**
14. **{**
15. entity**.**Update**(**gameTime**);**
16. **}**
17. **foreach** **(**IGameEntity entity **in** entitiesToRemove**)**
18. **{**
19. entities**.**Remove**(**entity**);**
20. **}**
21. entitiesToRemove**.**Clear**();**
22. **}**
23. **public** void Draw**(**SpriteBatch spriteBatch**,** GameTime gameTime**)**
24. **{**
25. **foreach** **(**IGameEntity entity **in** entities**.**OrderBy**(**e **=>** e**.**DrawOrder**))**
26. **{**
27. entity**.**Draw**(**spriteBatch**,** gameTime**);**
28. **}**
29. **}**
30. **public** void AddEntity**(**IGameEntity entity**)**
31. **{**
32. entities**.**Add**(**entity**);**
33. **}**
34. **public** void RemoveEntity**(**IGameEntity entity**)**
35. **{**
36. entitiesToRemove**.**Add**(**entity**);**
37. **}**
38. **public** void ClearEntities**()**
39. **{**
40. entities**.**Clear**();**
41. **}**
42. **}**
43. **}**

#### IGameEntity.cs

1. **using** Microsoft**.**Xna**.**Framework**;**
2. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
3. **namespace** Basic\_Wars\_V2**.**Entities
4. **{**
5. **public** **interface** IGameEntity
6. **{**
7. int DrawOrder **{** **get;** **}**
8. void Update**(**GameTime gameTime**);**
9. void Draw**(**SpriteBatch spriteBatch**,** GameTime gameTime**);**
10. **}**
11. **}**

#### ICollideable.cs

1. **using** Microsoft**.**Xna**.**Framework**;**
2. **namespace** Basic\_Wars\_V2**.**Entities
3. **{**
4. **public** **interface** ICollideable
5. **{**
6. Rectangle Collider **{** **get;** **}**
7. **}**
8. **}**

#### ButtonManager.cs

1. **using** Microsoft**.**Xna**.**Framework**;**
2. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
3. **using** System**.**Collections**.**Generic**;**
4. **namespace** Basic\_Wars\_V2**.**Entities
5. **{**
6. **public** class ButtonManager **:** IGameEntity
7. **{**
8. **public** List**<**Button**>** buttons **=** **new** List**<**Button**>();**
9. **public** int ID **=** 0**;**
10. **public** int DrawOrder **{** **get;** **set;** **}**
11. **public** void AddButton**(**Button button**)**
12. **{**
13. buttons**.**Add**(**button**);**
14. button**.**ID **=** ID**;**
15. ID**++;**
16. **}**
17. **public** void AddButtons**(**List**<**Button**>** buttons**)**
18. **{**
19. **foreach** **(**Button button **in** buttons**)**
20. **{**
21. AddButton**(**button**);**
22. **}**
23. **}**
24. **public** void DrawButtonIDs**(**int initialID **=** **-**1**,** int finalID **=** **-**1**,** int initial2ID **=** **-**1**,** int final2ID **=** **-**1**,** int initial3ID **=** **-**1**,** int final3ID **=** **-**1**)**
25. **{**
26. **foreach** **(**Button button **in** buttons**)**
27. **{**
28. **if** **((**button**.**ID **>=** initialID **&&** button**.**ID **<=** finalID**)** **||** **(**button**.**ID **>=** initial2ID **&&** button**.**ID **<=** final2ID**)** **||** **(**button**.**ID **>=** initial3ID **&&** button**.**ID **<=** final3ID**))**
29. **{**
30. button**.**DrawButton **=** **true;**
31. **}**
32. **else**
33. **{**
34. button**.**DrawButton **=** **false;**
35. **}**
36. **}**
37. **}**
38. **public** void UpdateButtonText**(**Button ButtonToChange**,** string Text**)**
39. **{**
40. **foreach** **(**Button button **in** buttons**)**
41. **{**
42. **if** **(**button **==** ButtonToChange**)**
43. **{**
44. button**.**UpdateButtonText**(**Text**);**
45. **}**
46. **}**
47. **}**
48. **public** void Update**(**GameTime gameTime**)**
49. **{**
50. **}**
51. **public** void Draw**(**SpriteBatch \_spriteBatch**,** GameTime gameTime**)**
52. **{**
53. **foreach** **(**Button button **in** buttons**)**
54. **{**
55. **if** **(**button**.**DrawButton**)**
56. **{**
57. button**.**Draw**(**\_spriteBatch**,** gameTime**);**
58. **}**
59. **}**
60. **}**
61. **}**
62. **}**

#### Button.cs

1. **using** Basic\_Wars**.**Graphics**;**
2. **using** Basic\_Wars\_V2**.**Graphics**;**
3. **using** Microsoft**.**Xna**.**Framework**;**
4. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
5. **namespace** Basic\_Wars\_V2**.**Entities
6. **{**
7. **public** class Button **:** IGameEntity**,** ICollideable
8. **{**
9. **private** const int X\_SPRITE\_SHEET\_START\_POS **=** 0**;**
10. **private** const int Y\_SPRITE\_SHEET\_START\_POS **=** 392**;**
11. **public** int DrawOrder **{** **get;** **set;** **}**
12. **private** Sprite buttonSprite**;**
13. **private** **readonly** Texture2D Texture**;**
14. **private** Vector2 ButtonPosition**;**
15. **private** Vector2 TextPosition**;**
16. **private** **readonly** SpriteFont spriteFont**;**
17. **private** Font text**;**
18. **private** int Width**;**
19. **private** int Height**;**
20. **public** int ID **{** **get;** **set;** **}**
21. **private** int ButtonType **{** **get;** **set;** **}**
22. **private** int ButtonShiftX **{** **get;** **set;** **}**
23. **private** int ButtonShiftY **{** **get;** **set;** **}**
24. **public** bool Pressed **{** **get;** **set;** **}**
25. **public** bool DrawButton **{** **get;** **set;** **}**
26. **public** Button**(**Texture2D texture**,** SpriteFont font**,** Vector2 position**,** int buttonType **=** **-**1**,** string Text **=** ""**)**
27. **{**
28. ButtonPosition **=** position**;**
29. Texture **=** texture**;**
30. spriteFont **=** font**;**
31. DrawButton **=** **false;**
32. ButtonType **=** buttonType**;**
33. CreateButtonSprite**();**
34. UpdateButtonText**(**Text**);**
35. Pressed **=** **false;**
36. **}**
37. **public** void CentreText**()**
38. **{**
39. TextPosition **=** **new** Vector2**(**ButtonPosition**.**X **+** Width **/** 2**,** ButtonPosition**.**Y **+** Height **/** 2**);**
40. **}**
41. **public** void CreateButtonSprite**()**
42. **{**
43. GetButtonType**();**
44. buttonSprite **=** **new** Sprite**(**Texture**,** X\_SPRITE\_SHEET\_START\_POS **+** ButtonShiftX**,** Y\_SPRITE\_SHEET\_START\_POS **+** ButtonShiftY**,** Width**,** Height**);**
45. **}**
46. **public** void GetButtonType**()**
47. **{**
48. **switch** **(**ButtonType**)**
49. **{**
50. **case** 0**:**
51. //Menu
52. ButtonShiftX **=** 0**;**
53. ButtonShiftY **=** 0**;**
54. Width **=** 672**;**
55. Height **=** 125**;**
56. **break;**
57. **case** 1**:**
58. //AltMenu
59. ButtonShiftX **=** 252**;**
60. ButtonShiftY **=** 127**;**
61. Width **=** 289**;**
62. Height **=** 125**;**
63. **break;**
64. **case** 2**:**
65. //Attribute
66. ButtonShiftX **=** 0**;**
67. ButtonShiftY **=** 127**;**
68. Width **=** 252**;**
69. Height **=** 378**;**
70. **break;**
71. **case** 3**:**
72. //Pause
73. ButtonShiftX **=** 252**;**
74. ButtonShiftY **=** 252**;**
75. Width **=** 131**;**
76. Height **=** 131**;**
77. **break;**
78. **case** 4**:**
79. //Tile
80. ButtonShiftX **=** 500**;**
81. ButtonShiftY **=** 500**;**
82. Width **=** 56**;**
83. Height **=** 56**;**
84. **break;**
85. **default:**
86. ButtonShiftX **=** 500**;**
87. ButtonShiftY **=** 500**;**
88. Width **=** 289**;**
89. Height **=** 127**;**
90. **break;**
91. **}**
92. **}**
93. **public** Rectangle Collider
94. **{**
95. **get**
96. **{**
97. **return** **new** Rectangle**((**int**)**ButtonPosition**.**X**,** **(**int**)**ButtonPosition**.**Y**,** Width**,** Height**);**
98. **}**
99. **}**
100. **public** void UpdateButtonText**(**string displayedText**)**
101. **{**
102. text **=** **new** Font**(**spriteFont**,** displayedText**);**
103. CentreText**();**
104. **}**
105. **public** void Draw**(**SpriteBatch \_spriteBatch**,** GameTime gameTime**)**
106. **{**
107. buttonSprite**.**Draw**(**\_spriteBatch**,** ButtonPosition**);**
108. text**.**WriteText**(**\_spriteBatch**,** TextPosition**);**
109. **}**
110. **public** void Update**(**GameTime gameTime**)**
111. **{**
112. **}**
113. **}**
114. **}**

#### UnitManager.cs

1. **using** Basic\_Wars\_V2**.**Enums**;**
2. **using** Microsoft**.**Xna**.**Framework**;**
3. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
4. **using** System**.**Collections**.**Generic**;**
5. **namespace** Basic\_Wars\_V2**.**Entities
6. **{**
7. **public** class UnitManager **:** IGameEntity
8. **{**
9. **public** List**<**Unit**>** units **=** **new** List**<**Unit**>();**
10. **public** List**<**Unit**>** unitsToRemove **=** **new** List**<**Unit**>();**
11. **public** bool DrawUnits **{** **get;** **set;** **}**
12. **public** int DrawOrder **=>** 1**;**
13. **public** UnitManager**()**
14. **{**
15. DrawUnits **=** **false;**
16. **}**
17. **public** void AddUnit**(**Unit unit**)**
18. **{**
19. units**.**Add**(**unit**);**
20. **}**
21. **public** void RemoveUnit**(**Unit unit**)**
22. **{**
23. unitsToRemove**.**Add**(**unit**);**
24. **}**
25. **public** List**<**Vector2**>** GetUnitPositions**()**
26. **{**
27. List**<**Vector2**>** positions **=** **new** List**<**Vector2**>();**
28. **foreach** **(**Unit unit **in** units**)**
29. **{**
30. positions**.**Add**(**unit**.**Position**);**
31. **}**
32. **return** positions**;**
33. **}**
34. **public** void ResetUnitStates**(**Player currentPlayer**)**
35. **{**
36. **foreach** **(**Unit unit **in** units**)**
37. **{**
38. **if** **(**unit**.**Team **==** currentPlayer**.**Team**)**
39. **{**
40. unit**.**State **=** UnitState**.**None**;**
41. **}**
42. **}**
43. **}**
44. **public** void Draw**(**SpriteBatch \_spriteBatch**,** GameTime gameTime**)**
45. **{**
46. **if** **(**DrawUnits**)**
47. **{**
48. **foreach** **(**Unit unit **in** units**)**
49. **{**
50. unit**.**Draw**(**\_spriteBatch**,** gameTime**);**
51. **}**
52. **}**
53. **}**
54. **public** void ClearUnits**()**
55. **{**
56. units**.**Clear**();**
57. **}**
58. **public** void Update**(**GameTime gameTime**)**
59. **{**
60. **foreach** **(**Unit unit **in** units**)**
61. **{**
62. unit**.**Update**(**gameTime**);**
63. **if** **(**unit**.**State **==** UnitState**.**Dead**)**
64. **{**
65. unitsToRemove**.**Add**(**unit**);**
66. **}**
67. **}**
68. **foreach** **(**Unit unit **in** unitsToRemove**)**
69. **{**
70. units**.**Remove**(**unit**);**
71. **}**
72. **}**
73. **}**
74. **}**

#### Unit.cs

1. **using** Basic\_Wars**.**Graphics**;**
2. **using** Basic\_Wars\_V2**.**Enums**;**
3. **using** Microsoft**.**Xna**.**Framework**;**
4. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
5. **namespace** Basic\_Wars\_V2**.**Entities
6. **{**
7. **public** class Unit **:** IGameEntity**,** ICollideable
8. **{**
9. **private** const int UNIT\_WIDTH **=** 56**;**
10. **private** const int UNIT\_HEIGHT **=** 56**;**
11. **private** const int X\_SPRITE\_SHEET\_START\_POS **=** 0**;**
12. **private** const int Y\_SPRITE\_SHEET\_START\_POS **=** 0**;**
13. //Animations not implemented
14. //private const int ANIMATION\_SHIFT = 56;
15. **private** const int SPRITE\_SHEET\_TEAM\_SHIFT **=** 168**;**
16. **private** const int SPRITE\_SHEET\_UNIT\_SHIFT **=** 56**;**
17. **private** Sprite unitSprite**;**
18. **private** Texture2D Texture **{** **get;** **set;** **}**
19. **public** Vector2 Position **{** **get;** **set;** **}**
20. **public** UnitState State **{** **get;** **set;** **}**
21. **public** int Team **{** **get;** **set;** **}**
22. **public** UnitType Type **{** **get;** **private** **set;** **}**
23. **public** int UnitTypeInt **{** **get;** **set;** **}**
24. **public** int Health **{** **get;** **set;** **}**
25. **public** int Ammo **{** **get;** **set;** **}**
26. **public** int Fuel **{** **get;** **set;** **}**
27. **public** float Defence **{** **get;** **set;** **}**
28. **public** int MovementPoints **{** **get;** **set;** **}**
29. **public** int CostToProduce **{** **get;** **set;** **}**
30. **public** bool Selected **{** **get;** **set;** **}**
31. **public** int DrawOrder **=>** 1**;**
32. **public** Unit**(**Texture2D texture**,** Vector2 position**,** int unitType**,** int team**)**
33. **{**
34. Texture **=** texture**;**
35. Position **=** position**;**
36. Team **=** team**;**
37. UnitTypeInt **=** unitType**;**
38. CreateUnitSprite**(**unitType**);**
39. SetStartingAttributes**();**
40. RefreshUnitAttributes**();**
41. **}**
42. **public** void CreateUnitSprite**(**int unitType**)**
43. **{**
44. int teamShift **=** **(**Team**)** **\*** SPRITE\_SHEET\_TEAM\_SHIFT**;**
45. int unitShift **=** **(**unitType **-** 1**)** **\*** SPRITE\_SHEET\_UNIT\_SHIFT**;**
46. unitSprite **=** **new** Sprite**(**Texture**,** X\_SPRITE\_SHEET\_START\_POS **+** teamShift**,** Y\_SPRITE\_SHEET\_START\_POS **+** unitShift**,** UNIT\_WIDTH**,** UNIT\_HEIGHT**);**
47. **}**
48. **public** Rectangle Collider
49. **{**
50. **get**
51. **{**
52. **return** **new** Rectangle**((**int**)**Position**.**X**,** **(**int**)**Position**.**Y**,** UNIT\_WIDTH**,** UNIT\_HEIGHT**);**
53. **}**
54. **}**
55. **public** void Draw**(**SpriteBatch \_spriteBatch**,** GameTime gameTime**)**
56. **{**
57. unitSprite**.**Draw**(**\_spriteBatch**,** Position**);**
58. **}**
59. **private** void SetStartingAttributes**()**
60. **{**
61. Selected **=** **false;**
62. Health **=** 100**;**
63. State **=** UnitState**.**None**;**
64. **}**
65. **public** void RefreshUnitAttributes**()**
66. **{**
67. **switch** **(**UnitTypeInt **-** 1**)**
68. **{**
69. **case** 0**:**
70. Type **=** UnitType**.**Infantry**;**
71. Ammo **=** 100**;**
72. Fuel **=** 50**;**
73. CostToProduce **=** 1000**;**
74. MovementPoints **=** 3**;**
75. **break;**
76. **case** 1**:**
77. Type **=** UnitType**.**Mech**;**
78. Ammo **=** 3**;**
79. Fuel **=** 40**;**
80. CostToProduce **=** 3000**;**
81. MovementPoints **=** 2**;**
82. **break;**
83. **case** 2**:**
84. Type **=** UnitType**.**Tank**;**
85. Ammo **=** 9**;**
86. Fuel **=** 25**;**
87. CostToProduce **=** 7000**;**
88. MovementPoints **=** 6**;**
89. **break;**
90. **case** 3**:**
91. Type **=** UnitType**.**APC**;**
92. Ammo **=** 0**;**
93. Fuel **=** 30**;**
94. CostToProduce **=** 5000**;**
95. MovementPoints **=** 6**;**
96. **break;**
97. **}**
98. **}**
99. **public** void Update**(**GameTime gameTime**)**
100. **{**
101. **if** **(**Health **<=** 0**)**
102. **{**
103. State **=** UnitState**.**Dead**;**
104. **}**
105. **}**
106. **}**
107. **}**

#### MapManager.cs

1. **using** Basic\_Wars\_V2**.**Enums**;**
2. **using** Basic\_Wars\_V2**.**System**;**
3. **using** Microsoft**.**Xna**.**Framework**;**
4. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
5. **using** System**;**
6. **using** System**.**Collections**.**Generic**;**
7. **namespace** Basic\_Wars\_V2**.**Entities
8. **{**
9. **public** class MapManager **:** IGameEntity**,** ICollideable
10. **{**
11. **private** const int WINDOW\_WIDTH **=** 1920**;**
12. **private** const int WINDOW\_HEIGHT **=** 1080**;**
13. **public** Tile**[,]** Map **{** **get;** **set;** **}**
14. **public** List**<**Tile**>** Structures **{** **get;** **private** **set;** **}** **=** **new** List**<**Tile**>();**
15. **public** List**<**Tile**>** HQs **{** **get;** **private** **set;** **}** **=** **new** List**<**Tile**>();**
16. **public** int MapWidth **{** **get;** **set;** **}**
17. **public** int MapHeight **{** **get;** **set;** **}**
18. **private** int NumOfPlayers **{** **get;** **set;** **}**
19. **public** bool DrawMap **{** **get;** **set;** **}**
20. **private** const int TILE\_DIMENSIONS **=** 56**;**
21. **public** Vector2 MapSize **{** **get;** **set;** **}**
22. **public** Vector2 Position **{** **get;** **set;** **}**
23. **private** Texture2D Texture **{** **get;** **set;** **}**
24. **public** int DrawOrder **=>** 0**;**
25. **private** int StructureSparsity **{** **get;** **set;** **}**
26. **public** MapManager**(**Texture2D texture**,** int mapWidth**,** int mapHeight**,** int numOfPlayers **=** 2**)**
27. **{**
28. MapWidth **=** mapWidth**;**
29. MapHeight **=** mapHeight**;**
30. Texture **=** texture**;**
31. NumOfPlayers **=** numOfPlayers**;**
32. Position **=** **new** Vector2**(**WINDOW\_WIDTH **/** 2 **-** **(**MapWidth **\*** TILE\_DIMENSIONS **/** 2**),** WINDOW\_HEIGHT **/** 2 **-** **(**MapHeight **\*** TILE\_DIMENSIONS **/** 2**));**
33. MapSize **=** **new** Vector2**(**mapWidth **\*** TILE\_DIMENSIONS**,** mapHeight **\*** TILE\_DIMENSIONS**);**
34. ResetMapSize**();**
35. StructureSparsity **=** **(**MapWidth **/** 6**)** **\*** TILE\_DIMENSIONS**;**
36. DrawMap **=** **false;**
37. GenerateMap**();**
38. **}**
39. **private** void GenerateMap**()**
40. **{**
41. GenerateBaseMap**();**
42. GenerateStructure**(**"City"**);**
43. GenerateStructure**(**"Factory"**);**
44. GenerateRoads**();**
45. GenerateHQs**();**
46. CreateTileSprites**();**
47. **}**
48. **public** void RegenerateMap**()**
49. **{**
50. GenerateRoads**();**
51. CreateTileSprites**();**
52. **}**
53. **private** void CreateTileSprites**()**
54. **{**
55. **foreach** **(**Tile tile **in** Map**)**
56. **{**
57. tile**.**CreateTileSpriteOnType**();**
58. **}**
59. **}**
60. **public** void ClearMap**()**
61. **{**
62. Array**.**Clear**(**Map**);**
63. **}**
64. **public** void ResetMapSize**()**
65. **{**
66. Map **=** **new** Tile**[**MapWidth**,** MapHeight**];**
67. **}**
68. **private** void GenerateBaseMap**()**
69. **{**
70. float x**;**
71. float y **=** Position**.**Y**;**
72. int randomTile**;**
73. Vector2 tempPosition**;**
74. **for** **(**int i **=** 0**;** i **<** MapHeight**;** i**++)**
75. **{**
76. x **=** Position**.**X**;**
77. **for** **(**int j **=** 0**;** j **<** MapWidth**;** j**++)**
78. **{**
79. randomTile **=** RandomTile**();**
80. tempPosition **=** **new** Vector2**(**x**,** y**);**
81. Tile newTile **=** **new(**tempPosition**,** Texture**)**
82. **{**
83. MapGridPos **=** **new** Vector2**(**j**,** i**)**
84. **};**
85. **switch** **(**randomTile**)**
86. **{**
87. **case** 0**:**
88. newTile**.**Type **=** TileType**.**Plains**;**
89. **break;**
90. **case** 1**:**
91. newTile**.**Type **=** TileType**.**Forest**;**
92. **break;**
93. **case** 2**:**
94. newTile**.**Type **=** TileType**.**Mountain**;**
95. **break;**
96. **}**
97. Map**[**j**,** i**]** **=** newTile**;**
98. x **+=** 56**;**
99. **}**
100. y **+=** 56**;**
101. **}**
102. **}**
103. **private** void GenerateStructure**(**string StructureType**)**
104. **{**
105. TileType Type **=** TileType**.**None**;**
106. List**<**Vector2**>** points**;**
107. **if** **(**StructureType **==** "City"**)**
108. **{**
109. Type **=** TileType**.**City**;**
110. **}**
111. **else** **if** **(**StructureType **==** "Factory"**)**
112. **{**
113. Type **=** TileType**.**Factory**;**
114. StructureSparsity **\*=** 3**;**
115. **}**
116. points **=** PoissonDiscSampling**.**GetPoints**(**StructureSparsity**,** MapSize**);**
117. **foreach** **(**Vector2 point **in** points**)**
118. **{**
119. int newGridX **=** **(**int**)(**point**.**X**)** **/** TILE\_DIMENSIONS**;**
120. int newGridY **=** **(**int**)(**point**.**Y**)** **/** TILE\_DIMENSIONS**;**
121. Vector2 newGridPos **=** **new(**Map**[**newGridX**,** newGridY**].**Position**.**X**,** Map**[**newGridX**,** newGridY**].**Position**.**Y**);**
122. Tile newStructure **=** **new(**newGridPos**,** Texture**);**
123. **if** **(!(**Map**[**newGridX**,** newGridY**].**Type **==** TileType**.**City**))**
124. **{**
125. newStructure**.**MapGridPos **=** **new** Vector2**(**newGridX**,** newGridY**);**
126. Map**[**newGridX**,** newGridY**]** **=** newStructure**;**
127. Map**[**newGridX**,** newGridY**].**Type **=** Type**;**
128. Structures**.**Add**(**newStructure**);**
129. **}**
130. **}**
131. **}**
132. **private** void GenerateRoads**()**
133. **{**
134. Vector2 firstStructureGridPos**;**
135. Vector2 nextStructureGridPos**;**
136. **for** **(**int i **=** 0**;** i **<** Structures**.**Count **-** 1**;** i **+=** 2**)**
137. **{**
138. firstStructureGridPos **=** Structures**[**i**].**MapGridPos**;**
139. nextStructureGridPos **=** Structures**[**i **+** 1**].**MapGridPos**;**
140. BuildRoad**(**firstStructureGridPos**,** nextStructureGridPos**);**
141. **}**
142. **}**
143. **private** void BuildRoad**(**Vector2 firstStrcuturePos**,** Vector2 nextStructurePos**)**
144. **{**
145. int x0 **=** **(**int**)**firstStrcuturePos**.**X**;**
146. int y0 **=** **(**int**)**firstStrcuturePos**.**Y**;**
147. int x1 **=** **(**int**)**nextStructurePos**.**X**;**
148. int y1 **=** **(**int**)**nextStructurePos**.**Y**;**
149. **while** **(**x0 **!=** x1**)**
150. **{**
151. **if** **(**x0 **>** x1**)**
152. **{**
153. x0**--;**
154. CreateRoadTile**(**x0**,** y0**,** 3**);**
155. **}**
156. **else** **if** **(**x0 **<** x1**)**
157. **{**
158. x0**++;**
159. CreateRoadTile**(**x0**,** y0**,** 3**);**
160. **}**
161. **}**
162. **while** **(**y0 **!=** y1**)**
163. **{**
164. **if** **(**y0 **>** y1**)**
165. **{**
166. y0**--;**
167. CreateRoadTile**(**x0**,** y0**,** 4**);**
168. **}**
169. **else** **if** **(**y0 **<** y1**)**
170. **{**
171. y0**++;**
172. CreateRoadTile**(**x0**,** y0**,** 4**);**
173. **}**
174. **}**
175. **}**
176. **private** void CreateRoadTile**(**int X**,** int Y**,** int direction**)**
177. **{**
178. **if** **(**Map**[**X**,** Y**].**Type **!=** TileType**.**City
179. **&&** Map**[**X**,** Y**].**Type **!=** TileType**.**Factory
180. **&&** Map**[**X**,** Y**].**Type **!=** TileType**.**Mountain
181. **&&** Map**[**X**,** Y**].**Type **!=** TileType**.**HQ
182. **)**
183. **{**
184. Tile roadTile **=** **new(**Map**[**X**,** Y**].**Position**,** Texture**)**
185. **{**
186. Type **=** TileType**.**Road**,**
187. MapGridPos **=** **new** Vector2**(**X**,** Y**)**
188. **};**
189. Map**[**X**,** Y**]** **=** roadTile**;**
190. Map**[**X**,** Y**].**CreateTileSprite**(**direction**);**
191. **}**
192. **}**
193. **public** void GenerateHQs**()**
194. **{**
195. int mapWidth **=** MapWidth **-** 1**;**
196. int mapHeight **=** MapHeight **-** 1**;**
197. **switch** **(**NumOfPlayers**)**
198. **{**
199. **case** 2**:**
200. CreateHQTile**(**0**,** 0**,** 0**);**
201. CreateHQTile**(**mapWidth**,** mapHeight**,** 1**);**
202. **break;**
203. **case** 3**:**
204. CreateHQTile**(**0**,** 0**,** 0**);**
205. CreateHQTile**(**mapWidth**,** mapHeight**,** 1**);**
206. CreateHQTile**(**mapWidth**,** 0**,** 2**);**
207. **break;**
208. **case** 4**:**
209. CreateHQTile**(**0**,** 0**,** 0**);**
210. CreateHQTile**(**mapWidth**,** mapHeight**,** 1**);**
211. CreateHQTile**(**mapWidth**,** 0**,** 2**);**
212. CreateHQTile**(**0**,** mapHeight**,** 3**);**
213. **break;**
214. **}**
215. **}**
216. **private** void CreateHQTile**(**int X**,** int Y**,** int team**)**
217. **{**
218. Tile HQTile **=** **new(**Map**[**X**,** Y**].**Position**,** Texture**)**
219. **{**
220. Type **=** TileType**.**HQ**,**
221. MapGridPos **=** **new** Vector2**(**X**,** Y**),**
222. Team **=** team
223. **};**
224. Map**[**X**,** Y**]** **=** HQTile**;**
225. Structures**.**Add**(**HQTile**);**
226. HQs**.**Add**(**HQTile**);**
227. **}**
228. **public** Rectangle Collider
229. **{**
230. **get**
231. **{**
232. **return** **new** Rectangle**((**int**)**Position**.**X**,** **(**int**)**Position**.**Y**,** MapWidth **\*** TILE\_DIMENSIONS**,** MapHeight **\*** TILE\_DIMENSIONS**);**
233. **}**
234. **}**
235. **private** static int RandomTile**()**
236. **{**
237. Random random **=** **new();**
238. int randTileNum **=** random**.**Next**(**0**,** 20**);**
239. **switch** **(**randTileNum**)**
240. **{**
241. **case** 0**:**
242. **case** 1**:**
243. **return** 1**;**
244. **case** 2**:**
245. **return** 2**;**
246. **default:**
247. **return** 0**;**
248. **}**
249. **}**
250. **public** int GetCost**(**Tile tile**,** Unit unit**)**
251. **{**
252. int terrainCost **=** 0**;**
253. **if** **(**unit**.**Type **==** UnitType**.**Infantry **||** unit**.**Type **==** UnitType**.**Mech**)**
254. **{**
255. **switch** **(**tile**.**Type**)**
256. **{**
257. **case** TileType**.**Plains**:**
258. **case** TileType**.**Forest**:**
259. **case** TileType**.**Road**:**
260. terrainCost **=** 1**;**
261. **break;**
262. **case** TileType**.**Mountain**:**
263. **case** TileType**.**City**:**
264. **case** TileType**.**Factory**:**
265. **case** TileType**.**HQ**:**
266. terrainCost **=** 2**;**
267. **break;**
268. **}**
269. **}**
270. **if** **(**unit**.**Type **==** UnitType**.**Tank **||** unit**.**Type **==** UnitType**.**APC**)**
271. **{**
272. **switch** **(**tile**.**Type**)**
273. **{**
274. **case** TileType**.**Road**:**
275. terrainCost **=** 1**;**
276. **break;**
277. **case** TileType**.**Plains**:**
278. **case** TileType**.**Forest**:**
279. terrainCost **=** 2**;**
280. **break;**
281. **case** TileType**.**City**:**
282. **case** TileType**.**Factory**:**
283. **case** TileType**.**HQ**:**
284. terrainCost **=** 3**;**
285. **break;**
286. **case** TileType**.**Mountain**:**
287. terrainCost **=** 6**;**
288. **break;**
289. **}**
290. **}**
291. **return** terrainCost**;**
292. **}**
293. **public** List**<**Tile**>** GetNeighbours**(**Tile tile**)**
294. **{**
295. List**<**Tile**>** neighbors **=** **new();**
296. int X **=** **(**int**)**tile**.**MapGridPos**.**X**;**
297. int Y **=** **(**int**)**tile**.**MapGridPos**.**Y**;**
298. **if** **(**X **>** 0**)**
299. **{**
300. neighbors**.**Add**(**Map**[**X **-** 1**,** Y**]);**
301. **}**
302. **if** **(**X **<** Map**.**GetLength**(**0**)** **-** 1**)**
303. **{**
304. neighbors**.**Add**(**Map**[**X **+** 1**,** Y**]);**
305. **}**
306. **if** **(**Y **>** 0**)**
307. **{**
308. neighbors**.**Add**(**Map**[**X**,** Y **-** 1**]);**
309. **}**
310. **if** **(**Y **<** Map**.**GetLength**(**1**)** **-** 1**)**
311. **{**
312. neighbors**.**Add**(**Map**[**X**,** Y **+** 1**]);**
313. **}**
314. **return** neighbors**;**
315. **}**
316. **public** void Draw**(**SpriteBatch spriteBatch**,** GameTime gameTime**)**
317. **{**
318. **if** **(**DrawMap**)**
319. **{**
320. **foreach** **(**Tile tile **in** Map**)**
321. **{**
322. tile**.**Draw**(**spriteBatch**,** gameTime**);**
323. **}**
324. **}**
325. **}**
326. **public** void Update**(**GameTime gameTime**)**
327. **{**
328. **}**
329. **}**
330. **}**

#### GameUI.cs

1. **using** Basic\_Wars\_V2**.**Enums**;**
2. **using** Basic\_Wars\_V2**.**System**;**
3. **using** Microsoft**.**Xna**.**Framework**;**
4. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
5. **using** System**.**Collections**.**Generic**;**
6. **namespace** Basic\_Wars\_V2**.**Entities
7. **{**
8. **public** class GameUI **:** IGameEntity
9. **{**
10. **private** **readonly** Dictionary**<(**UnitType**,** UnitType**),** int**>** baseDamageDictionary **=** **new()**
11. **{**
12. **{(**UnitType**.**Infantry**,** UnitType**.**Infantry**),** 55 **},**
13. **{(**UnitType**.**Infantry**,** UnitType**.**Mech**),** 40 **},**
14. **{(**UnitType**.**Infantry**,** UnitType**.**Tank**),** 10 **},**
15. **{(**UnitType**.**Infantry**,** UnitType**.**APC**),** 5 **},**
16. **{(**UnitType**.**Mech**,** UnitType**.**Infantry**),** 65 **},**
17. **{(**UnitType**.**Mech**,** UnitType**.**Mech**),** 55 **},**
18. **{(**UnitType**.**Mech**,** UnitType**.**Tank**),** 60 **},**
19. **{(**UnitType**.**Mech**,** UnitType**.**APC**),** 75 **},**
20. **{(**UnitType**.**Tank**,** UnitType**.**Infantry**),** 75 **},**
21. **{(**UnitType**.**Tank**,** UnitType**.**Mech**),** 70 **},**
22. **{(**UnitType**.**Tank**,** UnitType**.**Tank**),** 55 **},**
23. **{(**UnitType**.**Tank**,** UnitType**.**APC**),** 100 **},**
24. **};**
25. //Init
26. **private** Button BasicWarsTitle**;**
27. **private** Button NewGameButton**;**
28. **private** Button LoadGameButton**;**
29. **private** Button QuitGameButton**;**
30. //NewGame
31. **private** Button NumOfPlayersInfo**;**
32. **private** Button Players2Button**;**
33. **private** Button Players3Button**;**
34. **private** Button Players4Button**;**
35. **private** Button RefreshMapButton**;**
36. **private** Button StartGameButton**;**
37. **private** Button MenuButton**;**
38. //Turn
39. **private** Button TurnNumberInfo**;**
40. **private** Button CurrentPlayerTeamInfo**;**
41. **private** Button CurrentPlayerFundsInfo**;**
42. **private** Button EndTurnButton**;**
43. **private** Button PauseGameButton**;**
44. //DisplayActions
45. **private** Button PlayerIdleButton**;**
46. **private** Button PlayerMoveButton**;**
47. **private** Button PlayerAttackButton**;**
48. **private** Button ReturnButton**;**
49. **private** Button CaptureButton**;**
50. //DisplayAttributes
51. **private** Button AttributeDisplayInfo**;**
52. **private** Button TypeInfo**;**
53. **private** Button HealthInfo**;**
54. **private** Button AmmoInfo**;**
55. **private** Button FuelInfo**;**
56. **private** Button DefenceInfo**;**
57. //DisplayProductionChoices
58. **private** Button UnitInfantryButton**;**
59. **private** Button UnitMechButton**;**
60. **private** Button UnitTankButton**;**
61. **private** Button UnitAPCButton**;**
62. //Paused Game
63. **private** Button ResumeGameButton**;**
64. **private** Button SaveGameButton**;**
65. **private** Button MainMenuButton**;**
66. //Game Over Screen
67. **private** Button GameOverInfo**;**
68. **private** Button WinnerInfo**;**
69. **private** Button IncreaseMapSizeButton**;**
70. **private** Button DecreaseMapSizeButton**;**
71. **private** Button UnitResupplyButton**;**
72. **private** Button AddAIInfo**;**
73. **private** Button AIPlayerTrue**;**
74. **private** Button AIPlayerFalse**;**
75. **private** **readonly** Tile SelectedUI**;**
76. **public** bool DrawSelectedUI **{** **get;** **set;** **}**
77. **private** List**<**Tile**>** reachableTiles **=** **new();**
78. **private** **readonly** List**<**Tile**>** moveableOverlay **=** **new();**
79. **private** bool DrawReachable **=** **false;**
80. **private** **readonly** List**<**Tile**>** attackableTiles **=** **new();**
81. **private** **readonly** List**<**Tile**>** attackableOverlay **=** **new();**
82. **private** bool DrawAttackable **=** **false;**
83. **private** **readonly** List**<**Tile**>** tilesToBeRemoved **=** **new();**
84. **private** **readonly** Texture2D Texture**;**
85. **private** **readonly** SpriteFont Font**;**
86. **private** **readonly** ButtonManager \_buttonManager**;**
87. **private** **readonly** UnitManager \_unitManager**;**
88. **private** Dijkstra \_pathFinder**;**
89. **private** MapManager \_gameMap**;**
90. **private** int numOfplayers **=** 2**;**
91. **private** **readonly** int CentreButtonX **=** **(**1920 **-** 672**)** **/** 2**;**
92. **private** Player CurrentPlayer**;**
93. **private** Unit CurrentUnit**;**
94. **private** Tile CurrentTile**;**
95. **public** int DrawOrder **=>** 2**;**
96. **public** GameUI**(**Texture2D SpriteSheet**,** SpriteFont font**,** MapManager map**,** UnitManager unitManager**,** ButtonManager buttonManager**)**
97. **{**
98. Texture **=** SpriteSheet**;**
99. Font **=** font**;**
100. \_gameMap **=** map**;**
101. \_pathFinder **=** **new** Dijkstra**(**map**);**
102. \_unitManager **=** unitManager**;**
103. Player placeHolder **=** **new(**0**,** 0**);**
104. CurrentPlayer **=** placeHolder**;**
105. \_buttonManager **=** buttonManager**;**
106. InitialiseButtons**();**
107. DrawSelectedUI **=** **false;**
108. SelectedUI **=** **new** Tile**(new** Vector2**(**0**,** 0**),** Texture**);**
109. SelectedUI**.**CreateTileSprite**(**0**,** 1**);**
110. **}**
111. **public** void InitialiseButtons**()**
112. **{**
113. BasicWarsTitle **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 90**),** 0**,** "Basic Wars"**);**
114. NewGameButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 360**),** 0**,** "New Game"**);**
115. LoadGameButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 540**),** 0**,** "Load Game"**);**
116. QuitGameButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 720**),** 0**,** "Quit"**);**
117. NumOfPlayersInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX **+** 615**,** 0**),** 0**,** $"Number of Players: {numOfplayers}"**);**
118. Players2Button **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 180**),** 1**,** "2"**);**
119. Players3Button **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 304**),** 1**,** "3"**);**
120. Players4Button **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 428**),** 1**,** "4"**);**
121. RefreshMapButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 720**),** 1**,** "Refresh"**);**
122. StartGameButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 900**),** 0**,** "Start Game"**);**
123. MenuButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 0**),** 1**,** "Menu"**);**
124. EndTurnButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 925**),** 1**,** $"End Turn"**);**
125. TurnNumberInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 0**),** 1**,** $"Turn: 0"**);**
126. CurrentPlayerTeamInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**131**,** 0**),** 1**,** $"Player: {CurrentPlayer.Team}"**);**
127. CurrentPlayerFundsInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 126**),** 1**,** $"{CurrentPlayer.Funds}"**);**
128. PauseGameButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 0**),** 3**);**
129. PlayerIdleButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 300**),** 1**,** "Idle"**);**
130. PlayerMoveButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 425**),** 1**,** "Move"**);**
131. PlayerAttackButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 550**),** 1**,** "Attack"**);**
132. ReturnButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 925**),** 1**,** "Return"**);**
133. CaptureButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 675**),** 1**,** "Capture"**);**
134. AttributeDisplayInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 180**),** 2**);**
135. TypeInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 550**),** 1**);**
136. HealthInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**1625**,** 274.5f **-** 110**));**
137. AmmoInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**1625**,** 369 **-** 110**));**
138. FuelInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**1625**,** 463.5f **-** 110**));**
139. DefenceInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**1625**,** 558 **-** 110**));**
140. UnitInfantryButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 300**),** 1**,** "Infantry\n$1000"**);**
141. UnitMechButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 425**),** 1**,** "Mech\n$3000"**);**
142. UnitTankButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 550**),** 1**,** "Tank\n$7000"**);**
143. UnitAPCButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 675**),** 1**,** "APC\n$5000"**);**
144. ResumeGameButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 180**),** 0**,** "Resume"**);**
145. SaveGameButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 305**),** 0**,** "Save"**);**
146. MainMenuButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 430**),** 0**,** "Menu"**);**
147. GameOverInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 90**),** 0**,** "Game Over"**);**
148. WinnerInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**CentreButtonX**,** 215**),** 0**);**
149. IncreaseMapSizeButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 600**),** 1**,** "+"**);**
150. DecreaseMapSizeButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**1600**,** 725**),** 1**,** "-"**);**
151. UnitResupplyButton **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 800**),** 1**,** "Resupply"**);**
152. AddAIInfo **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 300**),** 1**,** "AI Player:\nFalse"**);**
153. AIPlayerTrue **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 425**),** 1**,** "True"**);**
154. AIPlayerFalse **=** **new(**Texture**,** Font**,** **new** Vector2**(**0**,** 550**),** 1**,** "False"**);**
155. \_buttonManager**.**AddButtons**(new** List**<**Button**>()**
156. **{**
157. BasicWarsTitle**,**
158. NewGameButton**,**
159. LoadGameButton**,**
160. QuitGameButton**,**
161. NumOfPlayersInfo**,**
162. Players2Button**,**
163. Players3Button**,**
164. Players4Button**,**
165. RefreshMapButton**,**
166. StartGameButton**,**
167. MenuButton**,**
168. TurnNumberInfo**,**
169. CurrentPlayerTeamInfo**,**
170. CurrentPlayerFundsInfo**,**
171. EndTurnButton**,**
172. PauseGameButton**,**
173. PlayerIdleButton**,**
174. PlayerMoveButton**,**
175. PlayerAttackButton**,**
176. ReturnButton**,**
177. CaptureButton**,**
178. AttributeDisplayInfo**,**
179. HealthInfo**,**
180. AmmoInfo**,**
181. FuelInfo**,**
182. DefenceInfo**,**
183. TypeInfo**,**
184. UnitInfantryButton**,**
185. UnitMechButton**,**
186. UnitTankButton**,**
187. UnitAPCButton**,**
188. ResumeGameButton**,**
189. SaveGameButton**,**
190. MainMenuButton**,**
191. GameOverInfo**,**
192. WinnerInfo**,**
193. IncreaseMapSizeButton**,**
194. DecreaseMapSizeButton**,**
195. UnitResupplyButton**,**
196. AddAIInfo**,**
197. AIPlayerTrue**,**
198. AIPlayerFalse**,**
199. **});**
200. **}**
201. **public** void ChangeSelectedPosition**(**Vector2 position**)**
202. **{**
203. SelectedUI**.**Position **=** position**;**
204. **}**
205. **public** List**<**Tile**>** GetReachableTiles**(**Unit unit**,** List**<**Vector2**>** UnitPositions**,** Tile StartingTile**)**
206. **{**
207. reachableTiles**.**Clear**();**
208. moveableOverlay**.**Clear**();**
209. reachableTiles **=** \_pathFinder**.**FindReachableTiles**(**StartingTile**,** unit**);**
210. **foreach** **(**Vector2 position **in** UnitPositions**)**
211. **{**
212. **foreach** **(**Tile tile **in** reachableTiles**)**
213. **{**
214. **if** **(**position **==** tile**.**Position **&&** position **!=** unit**.**Position**)**
215. **{**
216. tilesToBeRemoved**.**Add**(**tile**);**
217. **}**
218. **}**
219. **}**
220. **foreach** **(**Tile tile **in** tilesToBeRemoved**)**
221. **{**
222. reachableTiles**.**Remove**(**tile**);**
223. **}**
224. tilesToBeRemoved**.**Clear**();**
225. **foreach** **(**Tile tile **in** reachableTiles**)**
226. **{**
227. **if** **(**tile**.**Position **!=** unit**.**Position**)**
228. **{**
229. **if** **(**
230. **!(**tile**.**Type **==** TileType**.**Mountain
231. **&&** unit**.**Type **==** UnitType**.**Tank**)**
232. **&&** **!(**tile**.**Type **==** TileType**.**Mountain
233. **&&** unit**.**Type **==** UnitType**.**APC**)**
234. **)**
235. **{**
236. Tile overlayTile **=** **new(**tile**.**Position**,** Texture**);**
237. overlayTile**.**CreateTileSprite**(**2**,** 1**);**
238. moveableOverlay**.**Add**(**overlayTile**);**
239. **}**
240. **else**
241. **{**
242. tilesToBeRemoved**.**Add**(**tile**);**
243. **}**
244. **}**
245. **}**
246. //Blocking display of mountain tiles for vehicles
247. **foreach** **(**Tile tile **in** tilesToBeRemoved**)**
248. **{**
249. moveableOverlay**.**Remove**(**tile**);**
250. reachableTiles**.**Remove**(**tile**);**
251. **}**
252. tilesToBeRemoved**.**Clear**();**
253. **return** reachableTiles**;**
254. **}**
255. **public** List**<**Tile**>** GetAttackableTiles**(**Unit attackingUnit**,** Tile StartingTile**)**
256. **{**
257. attackableTiles**.**Clear**();**
258. attackableOverlay**.**Clear**();**
259. List**<**Tile**>** adjacentTiles **=** \_gameMap**.**GetNeighbours**(**StartingTile**);**
260. **foreach** **(**Tile tile **in** adjacentTiles**)**
261. **{**
262. **foreach** **(**Unit defendingUnit **in** \_unitManager**.**units**)**
263. **{**
264. **if** **(**defendingUnit**.**Position **==** tile**.**Position
265. **&&** defendingUnit**.**Team **!=** attackingUnit**.**Team
266. **)**
267. **{**
268. Tile overlayTile **=** **new(**tile**.**Position**,** Texture**);**
269. overlayTile**.**CreateTileSprite**(**1**,** 1**);**
270. attackableOverlay**.**Add**(**overlayTile**);**
271. attackableTiles**.**Add**(**tile**);**
272. **}**
273. **}**
274. **}**
275. **return** attackableTiles**;**
276. **}**
277. **public** void ChangeMap**(**MapManager gameMap**)**
278. **{**
279. \_gameMap **=** gameMap**;**
280. \_pathFinder **=** **new** Dijkstra**(**gameMap**);**
281. **}**
282. **public** void Update**(**GameTime gameTime**)**
283. **{**
284. **}**
285. **public** void Draw**(**SpriteBatch \_spriteBatch**,** GameTime gameTime**)**
286. **{**
287. **if** **(**DrawSelectedUI**)**
288. **{**
289. SelectedUI**.**Draw**(**\_spriteBatch**,** gameTime**);**
290. **}**
291. **if** **(**DrawReachable**)**
292. **{**
293. **foreach** **(**Tile tile **in** moveableOverlay**)**
294. **{**
295. tile**.**Draw**(**\_spriteBatch**,** gameTime**);**
296. **}**
297. **}**
298. **if** **(**DrawAttackable**)**
299. **{**
300. **foreach** **(**Tile tile **in** attackableOverlay**)**
301. **{**
302. tile**.**Draw**(**\_spriteBatch**,** gameTime**);**
303. **}**
304. **}**
305. \_buttonManager**.**Draw**(**\_spriteBatch**,** gameTime**);**
306. **}**
307. **public** MenuState Init**(**GameTime gameTime**,** Button PressedButton**)**
308. **{**
309. \_buttonManager**.**DrawButtonIDs**(**0**,** 3**);**
310. **if** **(**PressedButton **!=** **null)**
311. **{**
312. **switch** **(**PressedButton**.**ID**)**
313. **{**
314. **case** 1**:**
315. **return** MenuState**.**RefreshMap**;**
316. **case** 2**:**
317. **return** MenuState**.**LoadGame**;**
318. **case** 3**:**
319. **return** MenuState**.**QuitGame**;**
320. **}**
321. **}**
322. **return** MenuState**.**Initial**;**
323. **}**
324. **public** MenuState NewGame**(**GameTime gameTime**,** Button PressedButton**)**
325. **{**
326. \_buttonManager**.**UpdateButtonText**(**CurrentPlayerTeamInfo**,** ""**);**
327. \_buttonManager**.**UpdateButtonText**(**CurrentPlayerFundsInfo**,** ""**);**
328. \_buttonManager**.**UpdateButtonText**(**TurnNumberInfo**,** ""**);**
329. \_buttonManager**.**DrawButtonIDs**(**4**,** 10**,** 36**,** 37**,** 39**,** 41**);**
330. **if** **(**PressedButton **!=** **null)**
331. **{**
332. **switch** **(**PressedButton**.**ID**)**
333. **{**
334. **case** 5**:**
335. numOfplayers **=** 2**;**
336. \_buttonManager**.**UpdateButtonText**(**NumOfPlayersInfo**,** $"Number of Players: {numOfplayers}"**);**
337. **return** MenuState**.**RefreshMap**;**
338. **case** 6**:**
339. numOfplayers **=** 3**;**
340. \_buttonManager**.**UpdateButtonText**(**NumOfPlayersInfo**,** $"Number of Players: {numOfplayers}"**);**
341. **return** MenuState**.**RefreshMap**;**
342. **case** 7**:**
343. numOfplayers **=** 4**;**
344. \_buttonManager**.**UpdateButtonText**(**NumOfPlayersInfo**,** $"Number of Players: {numOfplayers}"**);**
345. **return** MenuState**.**RefreshMap**;**
346. **case** 8**:**
347. **return** MenuState**.**RefreshMap**;**
348. **case** 9**:**
349. **return** MenuState**.**PlayingGame**;**
350. **case** 10**:**
351. **return** MenuState**.**Initial**;**
352. **case** 36**:**
353. **return** MenuState**.**IncreaseMapSize**;**
354. **case** 37**:**
355. **return** MenuState**.**DecreaseMapSize**;**
356. **case** 40**:**
357. \_buttonManager**.**UpdateButtonText**(**AddAIInfo**,** "AI Player:\nTrue"**);**
358. **return** MenuState**.**AddAI**;**
359. **case** 41**:**
360. \_buttonManager**.**UpdateButtonText**(**AddAIInfo**,** "AI Player:\nFalse"**);**
361. **return** MenuState**.**RemoveAI**;**
362. **}**
363. **}**
364. **return** MenuState**.**NewGame**;**
365. **}**
366. **public** List**<**Player**>** GetPlayers**()**
367. **{**
368. List**<**Player**>** Players **=** **new();**
369. **for** **(**int i **=** 0**;** i **<** numOfplayers**;** i**++)**
370. **{**
371. Player player **=** **new(**i**,** 0**);**
372. Players**.**Add**(**player**);**
373. **}**
374. **return** Players**;**
375. **}**
376. **public** GameState Turn**(**GameTime gameTime**,** Player currentPlayer**,** int turnNumber**,** Button PressedButton**)**
377. **{**
378. **if** **(**CurrentUnit **!=** **null** **||** CurrentTile **!=** **null)**
379. **{**
380. \_buttonManager**.**DrawButtonIDs**(**11**,** 15**,** 21**,** 26**);**
381. **}**
382. **else**
383. **{**
384. \_buttonManager**.**DrawButtonIDs**(**11**,** 15**);**
385. **}**
386. CurrentPlayer **=** currentPlayer**;**
387. \_buttonManager**.**UpdateButtonText**(**CurrentPlayerTeamInfo**,** $"Team\n{currentPlayer.Colour}"**);**
388. \_buttonManager**.**UpdateButtonText**(**CurrentPlayerFundsInfo**,** $"${currentPlayer.Funds}"**);**
389. \_buttonManager**.**UpdateButtonText**(**TurnNumberInfo**,** $"Turn: {turnNumber}"**);**
390. **if** **(**PressedButton **!=** **null)**
391. **{**
392. **if** **(**PressedButton**.**ID **==** 14**)**
393. **{**
394. **return** GameState**.**EnemyTurn**;**
395. **}**
396. **if** **(**PressedButton**.**ID **==** 15**)**
397. **{**
398. **return** GameState**.**PauseGame**;**
399. **}**
400. **}**
401. **return** GameState**.**PlayerSelect**;**
402. **}**
403. **public** GameState DisplayPlayerActions**(**GameTime gameTime**,** Button PressedButton**,** bool displayCapture **=** **false,** bool displayResupply **=** **false)**
404. **{**
405. **if** **(**displayCapture **&&** displayResupply**)**
406. **{**
407. \_buttonManager**.**DrawButtonIDs**(**11**,** 20**,** 21**,** 26**,** 38**,** 38**);**
408. **}**
409. **else** **if** **(**displayCapture **&&** **!**displayResupply**)**
410. **{**
411. \_buttonManager**.**DrawButtonIDs**(**11**,** 20**,** 21**,** 26**);**
412. **}**
413. **else** **if** **(!**displayCapture **&&** displayResupply**)**
414. **{**
415. \_buttonManager**.**DrawButtonIDs**(**11**,** 19**,** 21**,** 26**,** 38**,** 38**);**
416. **}**
417. **else**
418. **{**
419. \_buttonManager**.**DrawButtonIDs**(**11**,** 19**,** 21**,** 26**);**
420. **}**
421. **if** **(**PressedButton **!=** **null)**
422. **{**
423. **switch** **(**PressedButton**.**ID**)**
424. **{**
425. **case** 16**:**
426. **return** GameState**.**UnitIdle**;**
427. **case** 17**:**
428. DrawReachable **=** **true;**
429. **return** GameState**.**PlayerMove**;**
430. **case** 18**:**
431. DrawAttackable **=** **true;**
432. **return** GameState**.**PlayerAttack**;**
433. **case** 19**:**
434. DrawReachable **=** **false;**
435. DrawAttackable **=** **false;**
436. **return** GameState**.**PlayerSelect**;**
437. **case** 38**:**
438. **return** GameState**.**PlayerResupply**;**
439. **}**
440. **if** **(**displayCapture**)**
441. **{**
442. **if** **(**PressedButton**.**ID **==** 20**)**
443. **{**
444. **return** GameState**.**PlayerCapture**;**
445. **}**
446. **}**
447. **}**
448. DrawReachable **=** **false;**
449. DrawAttackable **=** **false;**
450. **return** GameState**.**SelectAction**;**
451. **}**
452. **public** void DisplayAttributes**(**Unit unit **=** **null,** Tile tile **=** **null)**
453. **{**
454. \_buttonManager**.**UpdateButtonText**(**TypeInfo**,** ""**);**
455. \_buttonManager**.**UpdateButtonText**(**HealthInfo**,** ""**);**
456. \_buttonManager**.**UpdateButtonText**(**AmmoInfo**,** ""**);**
457. \_buttonManager**.**UpdateButtonText**(**FuelInfo**,** ""**);**
458. \_buttonManager**.**UpdateButtonText**(**DefenceInfo**,** ""**);**
459. CurrentUnit **=** unit**;**
460. CurrentTile **=** tile**;**
461. **if** **(**CurrentUnit **!=** **null** **||** CurrentTile **!=** **null)**
462. **{**
463. \_buttonManager**.**DrawButtonIDs**(**11**,** 15**,** 24**,** 26**);**
464. **if** **(**CurrentUnit **!=** **null)**
465. **{**
466. \_buttonManager**.**UpdateButtonText**(**TypeInfo**,** $"{CurrentUnit.Type}"**);**
467. \_buttonManager**.**UpdateButtonText**(**HealthInfo**,** $"{CurrentUnit.Health}"**);**
468. \_buttonManager**.**UpdateButtonText**(**AmmoInfo**,** $"{CurrentUnit.Ammo}"**);**
469. \_buttonManager**.**UpdateButtonText**(**FuelInfo**,** $"{CurrentUnit.Fuel}"**);**
470. \_buttonManager**.**UpdateButtonText**(**DefenceInfo**,** $"{CurrentUnit.Defence}%"**);**
471. **}**
472. **if** **(**CurrentTile **!=** **null)**
473. **{**
474. \_buttonManager**.**UpdateButtonText**(**TypeInfo**,** $"{CurrentTile.Type}"**);**
475. \_buttonManager**.**UpdateButtonText**(**DefenceInfo**,** $"{CurrentTile.DefenceBonus}%"**);**
476. **}**
477. **}**
478. **}**
479. **public** int ProcessUnitProduction**(**GameTime gameTime**,** Button PressedButton**)**
480. **{**
481. \_buttonManager**.**DrawButtonIDs**(**11**,** 15**,** 19**,** 19**,** 27**,** 30**);**
482. **if** **(**PressedButton **!=** **null)**
483. **{**
484. **switch** **(**PressedButton**.**ID**)**
485. **{**
486. **case** 19**:**
487. **return** **-**1**;**
488. **case** 27**:**
489. **return** 1**;**
490. **case** 28**:**
491. **return** 2**;**
492. **case** 29**:**
493. **return** 3**;**
494. **case** 30**:**
495. **return** 4**;**
496. **}**
497. **}**
498. **return** **-**2**;**
499. **}**
500. **public** MenuState PausedGame**(**GameTime gameTime**,** Button PressedButton**)**
501. **{**
502. \_buttonManager**.**DrawButtonIDs**(**31**,** 33**,** 3**,** 3**);**
503. **if** **(**PressedButton **!=** **null)**
504. **{**
505. **switch** **(**PressedButton**.**ID**)**
506. **{**
507. **case** 3**:**
508. **return** MenuState**.**QuitGame**;**
509. **case** 31**:**
510. **return** MenuState**.**PlayingGame**;**
511. **case** 32**:**
512. **return** MenuState**.**SaveGame**;**
513. **case** 33**:**
514. **return** MenuState**.**Initial**;**
515. **}**
516. **}**
517. **return** MenuState**.**GamePaused**;**
518. **}**
519. **public** MenuState GameOver**(**GameTime gameTime**,** Button PressedButton**,** Player Winner**)**
520. **{**
521. \_buttonManager**.**UpdateButtonText**(**WinnerInfo**,** $"Winner: {Winner.Colour} Team"**);**
522. \_buttonManager**.**DrawButtonIDs**(**3**,** 3**,** 33**,** 35**);**
523. **if** **(**PressedButton **!=** **null)**
524. **{**
525. **switch** **(**PressedButton**.**ID**)**
526. **{**
527. **case** 3**:**
528. **return** MenuState**.**QuitGame**;**
529. **case** 33**:**
530. **return** MenuState**.**Initial**;**
531. **}**
532. **}**
533. **return** MenuState**.**GameOver**;**
534. **}**
535. **public** void ClearAttackableOverlay**()**
536. **{**
537. attackableOverlay**.**Clear**();**
538. **}**
539. **public** void ClearMoveableOverlay**()**
540. **{**
541. moveableOverlay**.**Clear**();**
542. **}**
543. **public** int CalculateDamage**(**Unit attackingUnit**,** Unit defendingUnit**)**
544. **{**
545. int baseDamage **=** baseDamageDictionary**[(**attackingUnit**.**Type**,** defendingUnit**.**Type**)];**
546. double defenceMultiplier **=** **(**double**)(**defendingUnit**.**Defence**)** **/** 100**;**
547. double HealthMultiplier **=** **(**double**)**attackingUnit**.**Health **/** 100**;**
548. **return** **(**int**)(**HealthMultiplier **\*** **(**baseDamage **-** **(**baseDamage **\*** defenceMultiplier**)));**
549. **}**
550. **}**
551. **}**

#### Tile.cs

1. **using** Basic\_Wars**.**Graphics**;**
2. **using** Basic\_Wars\_V2**.**Enums**;**
3. **using** Microsoft**.**Xna**.**Framework**;**
4. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
5. **namespace** Basic\_Wars\_V2**.**Entities
6. **{**
7. **public** class Tile **:** IGameEntity**,** ICollideable
8. **{**
9. **private** const int TILE\_WIDTH **=** 56**;**
10. **private** const int TILE\_HEIGHT **=** 56**;**
11. **private** const int X\_SPRITE\_SHEET\_START\_POS **=** 336**;**
12. **private** const int Y\_SPRITE\_SHEET\_START\_POS **=** 224**;**
13. **public** TileType Type **{** **get;** **set;** **}**
14. **public** Vector2 Position **{** **get;** **set;** **}**
15. **public** Vector2 MapGridPos **{** **get;** **set;** **}**
16. **public** int TotalCost **{** **get;** **set;** **}**
17. **private** Texture2D Texture **{** **get;** **set;** **}**
18. **public** Sprite TileSprite **{** **get;** **set;** **}**
19. **public** int Team **{** **get;** **set;** **}**
20. **public** int DrawOrder **{** **get;** **set;** **}**
21. **public** int DefenceBonus **{** **get;** **set;** **}**
22. **public** bool Selected **{** **get;** **set;** **}**
23. **public** Tile**(**Vector2 position**,** Texture2D texture**,** int team **=** **-**1**)**
24. **{**
25. Position **=** position**;**
26. Texture **=** texture**;**
27. Team **=** team**;**
28. Selected **=** **false;**
29. **}**
30. **public** void CreateTileSprite**(**int TileColumn **=** 0**,** int TileRow **=** 0**)**
31. **{**
32. TileColumn **=** TileColumn **\*** TILE\_WIDTH**;**
33. TileRow **=** TileRow **\*** TILE\_HEIGHT**;**
34. TileSprite **=** **new** Sprite**(**Texture**,** X\_SPRITE\_SHEET\_START\_POS **+** TileColumn**,** Y\_SPRITE\_SHEET\_START\_POS **+** TileRow**,** TILE\_WIDTH**,** TILE\_HEIGHT**);**
35. SetTileAttributes**();**
36. **}**
37. **public** void CreateTileSpriteOnType**()**
38. **{**
39. **switch** **(**Type**)**
40. **{**
41. **case** TileType**.**Plains**:**
42. CreateTileSprite**();**
43. **break;**
44. **case** TileType**.**Forest**:**
45. CreateTileSprite**(**1**,** 0**);**
46. **break;**
47. **case** TileType**.**Mountain**:**
48. CreateTileSprite**(**2**,** 0**);**
49. **break;**
50. **case** TileType**.**City**:**
51. CreateTileSprite**(-**5 **+** Team**,** 0**);**
52. **break;**
53. **case** TileType**.**Factory**:**
54. CreateTileSprite**(-**5 **+** Team**,** 1**);**
55. **break;**
56. **case** TileType**.**HQ**:**
57. CreateTileSprite**(-**5 **+** Team**,** 2**);**
58. **break;**
59. **}**
60. **}**
61. **public** Rectangle Collider
62. **{**
63. **get**
64. **{**
65. **return** **new** Rectangle**((**int**)**Position**.**X**,** **(**int**)**Position**.**Y**,** TILE\_WIDTH**,** TILE\_HEIGHT**);**
66. **}**
67. **}**
68. **public** void SetTileAttributes**()**
69. **{**
70. **switch** **(**Type**)**
71. **{**
72. **case** TileType**.**Plains**:**
73. DefenceBonus **=** 10**;**
74. **break;**
75. **case** TileType**.**Forest**:**
76. DefenceBonus **=** 20**;**
77. **break;**
78. **case** TileType**.**Mountain**:**
79. DefenceBonus **=** 40**;**
80. **break;**
81. **case** TileType**.**Road**:**
82. DefenceBonus **=** 0**;**
83. **break;**
84. **case** TileType**.**City**:**
85. DefenceBonus **=** 20**;**
86. **break;**
87. **case** TileType**.**Factory**:**
88. DefenceBonus **=** 20**;**
89. **break;**
90. **case** TileType**.**HQ**:**
91. DefenceBonus **=** 20**;**
92. **break;**
93. **}**
94. **}**
95. **public** void Draw**(**SpriteBatch spriteBatch**,** GameTime gameTime**)**
96. **{**
97. TileSprite**.**Draw**(**spriteBatch**,** Position**);**
98. **}**
99. **public** void Update**(**GameTime gameTime**)**
100. **{**
101. **}**
102. **}**
103. **}**

#### Player.cs

1. **namespace** Basic\_Wars\_V2**.**Entities
2. **{**
3. **public** class Player
4. **{**
5. **public** int Team **{** **get;** **set;** **}**
6. **public** int Funds **{** **get;** **set;** **}**
7. **public** bool HasHQ **{** **get;** **set;** **}**
8. **public** string Colour **{** **get;** **set;** **}**
9. **public** bool IsAI **{** **get;** **set;** **}**
10. **public** Player**(**int team**,** int initialFunds**,** bool isAI **=** **false)**
11. **{**
12. Team **=** team**;**
13. Funds **=** initialFunds**;**
14. HasHQ **=** **true;**
15. IsAI **=** isAI**;**
16. GetTeamColour**();**
17. **}**
18. **private** void GetTeamColour**()**
19. **{**
20. **switch** **(**Team**)**
21. **{**
22. **case** 0**:**
23. Colour **=** "Red"**;**
24. **break;**
25. **case** 1**:**
26. Colour **=** "Blue"**;**
27. **break;**
28. **case** 2**:**
29. Colour **=** "Green"**;**
30. **break;**
31. **case** 3**:**
32. Colour **=** "Yellow"**;**
33. **break;**
34. **}**
35. **}**
36. **}**
37. **}**

### Enums

#### AIState.cs

1. **namespace** Basic\_Wars\_V2**.**Enums
2. **{**
3. **public** enum AIState
4. **{**
5. Initial**,**
6. Attack**,**
7. Defend**,**
8. CaptureStructures**,**
9. **}**
10. **}**

#### GameState.cs

1. **namespace** Basic\_Wars\_V2**.**Enums
2. **{**
3. **public** enum GameState
4. **{**
5. PlayerSelect**,**
6. SelectAction**,**
7. UnitIdle**,**
8. PlayerMove**,**
9. PlayerAttack**,**
10. PlayerCapture**,**
11. PlayerResupply**,**
12. PlayerProduceUnit**,**
13. EnemyTurn**,**
14. AITurn**,**
15. PauseGame**,**
16. **}**
17. **}**

#### MenuState.cs

1. **namespace** Basic\_Wars\_V2**.**Enums
2. **{**
3. **public** enum MenuState
4. **{**
5. Initial**,**
6. NewGame**,**
7. AddAI**,**
8. RemoveAI**,**
9. IncreaseMapSize**,**
10. DecreaseMapSize**,**
11. SaveGame**,**
12. LoadGame**,**
13. RefreshMap**,**
14. PlayingGame**,**
15. GameOver**,**
16. GamePaused**,**
17. QuitGame**,**
18. **}**
19. **}**

#### TileType.cs

1. **namespace** Basic\_Wars\_V2**.**Enums
2. **{**
3. **public** enum TileType
4. **{**
5. None**,**
6. Plains**,**
7. Forest**,**
8. Mountain**,**
9. Road**,**
10. City**,**
11. Factory**,**
12. HQ**,**
13. **}**
14. **}**

#### UnitState.cs

1. **namespace** Basic\_Wars\_V2**.**Enums
2. **{**
3. **public** enum UnitState
4. **{**
5. None**,**
6. Moved**,**
7. Used**,**
8. Dead**,**
9. **}**
10. **}**

#### UnitType.cs

1. **namespace** Basic\_Wars\_V2**.**Enums
2. **{**
3. **public** enum UnitType
4. **{**
5. Infantry**,**
6. Mech**,**
7. Tank**,**
8. APC**,**
9. **}**
10. **}**

### System

#### AI.cs

1. **using** Basic\_Wars\_V2**.**Entities**;**
2. **using** Basic\_Wars\_V2**.**Enums**;**
3. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
4. **using** System**;**
5. **using** System**.**Collections**.**Generic**;**
6. **namespace** Basic\_Wars\_V2**.**System
7. **{**
8. **public** class AI **:** Player
9. **{**
10. **private** **readonly** Random random **=** **new();**
11. **private** Texture2D Texture **{** **get;** **set;** **}**
12. **private** MapManager \_gameMap **{** **get;** **set;** **}**
13. **private** UnitManager \_unitManager **{** **get;** **set;** **}**
14. **private** GameUI \_gameUI **{** **get;** **set;** **}**
15. **private** InputController \_inputController **{** **get;** **set;** **}**
16. **private** Tile HQ **{** **get;** **set;** **}**
17. **private** **readonly** List**<**Tile**>** FriendlyBuildings **=** **new();**
18. **private** **readonly** List**<**Tile**>** EnemyHQs **=** **new();**
19. **private** **readonly** List**<**Unit**>** AIUnits **=** **new();**
20. **private** Tile HQToAttack**;**
21. **private** List**<**Tile**>** reachableTiles **=** **new();**
22. **private** **readonly** List**<**Tile**>** closeByStructures **=** **new();**
23. **private** AIState State **{** **get;** **set;** **}**
24. **public** AI**(**int team**,** int initialFunds**,** MapManager map**,** UnitManager unitManager**,** GameUI gameUI**,** InputController inputController**,** Texture2D texture**)** **:** **base(**team**,** initialFunds**)**
25. **{**
26. State **=** AIState**.**Initial**;**
27. \_gameMap **=** map**;**
28. \_unitManager **=** unitManager**;**
29. \_gameUI **=** gameUI**;**
30. \_inputController **=** inputController**;**
31. Texture **=** texture**;**
32. **}**
33. **public** void RefreshAI**(**MapManager map**,** UnitManager unitManager**)**
34. **{**
35. \_gameMap **=** map**;**
36. \_unitManager **=** unitManager**;**
37. **}**
38. **public** void RunAILogic**()**
39. **{**
40. GetEnemyHQs**();**
41. GetAIUnits**();**
42. **switch** **(**State**)**
43. **{**
44. **case** AIState**.**Initial**:**
45. ProduceUnits**();**
46. **break;**
47. **case** AIState**.**Attack**:**
48. **foreach** **(**Unit unit **in** AIUnits**)**
49. **{**
50. **if** **(**unit**.**State **!=** UnitState**.**Moved
51. **&&** unit**.**State **!=** UnitState**.**Used
52. **&&** unit**.**Fuel **>** 0
53. **)**
54. **{**
55. **if** **(**unit**.**Type **!=** UnitType**.**Tank
56. **&&** unit**.**Type **!=** UnitType**.**APC
57. **)**
58. **{**
59. MoveTowardsTile**(**unit**,** HQToAttack**);**
60. **}**
61. **else**
62. **{**
63. MoveToRandomReachable**(**unit**,** **true);**
64. **}**
65. **}**
66. **}**
67. **break;**
68. **case** AIState**.**Defend**:**
69. **foreach** **(**Unit unit **in** AIUnits**)**
70. **{**
71. **if** **(**unit**.**State **!=** UnitState**.**Moved
72. **&&** unit**.**State **!=** UnitState**.**Used
73. **&&** unit**.**Fuel **>** 0
74. **)**
75. **{**
76. MoveTowardsTile**(**unit**,** HQ**);**
77. **}**
78. **}**
79. **break;**
80. **case** AIState**.**CaptureStructures**:**
81. **foreach** **(**Unit unit **in** AIUnits**)**
82. **{**
83. reachableTiles **=** GetReachableTiles**(**unit**);**
84. **foreach** **(**Tile structure **in** closeByStructures**)**
85. **{**
86. **if** **(**reachableTiles**.**Contains**(**structure**)**
87. **&&** structure**.**Team **!=** Team
88. **&&** unit**.**State **!=** UnitState**.**Moved
89. **&&** unit**.**State **!=** UnitState**.**Used
90. **&&** unit**.**Type **!=** UnitType**.**Tank
91. **&&** unit**.**Type **!=** UnitType**.**APC
92. **&&** unit**.**Fuel **>** 0
93. **)**
94. **{**
95. MoveTowardsTile**(**unit**,** structure**);**
96. **}**
97. **else** **if** **(**unit**.**State **!=** UnitState**.**Moved
98. **&&** unit**.**State **!=** UnitState**.**Used
99. **&&** unit**.**Fuel **>** 0
100. **)**
101. **{**
102. MoveToRandomReachable**(**unit**,** **true);**
103. **}**
104. **}**
105. **}**
106. **break;**
107. **}**
108. AttackNeighbouringUnits**();**
109. CaptureStructures**();**
110. ProduceUnits**();**
111. //Setting State
112. **if** **(**EnemyCloseToHQ**())** //Prioritises its own HQ
113. **{**
114. State **=** AIState**.**Defend**;**
115. **}**
116. **else if (**Winning**())** //Attacks when in an advantageous position
117. **{**
118. State **=** AIState**.**Attack**;**
119. **}**
120. **else** **if** **(**CloseByStructure**())** //Moves towards structures if in range of its units
121. **{**
122. State **=** AIState**.**CaptureStructures**;**
123. **}**
124. **}**
125. **private** void CaptureStructures**()**
126. **{**
127. **foreach** **(**Unit unit **in** AIUnits**)**
128. **{**
129. Tile unitTile **=** \_inputController**.**GetUnitTile**(**unit**);**
130. **if** **((**unitTile**.**Type **==** TileType**.**City
131. **||** unitTile**.**Type **==** TileType**.**Factory
132. **||** unitTile**.**Type **==** TileType**.**HQ**)**
133. **&&** unit**.**Type **!=** UnitType**.**Tank
134. **&&** unit**.**Type **!=** UnitType**.**APC
135. **&&** unitTile**.**Team **!=** Team
136. **&&** unit**.**State **!=** UnitState**.**Used
137. **)**
138. **{**
139. unitTile**.**Team **=** Team**;**
140. unitTile**.**CreateTileSpriteOnType**();**
141. unit**.**State **=** UnitState**.**Used**;**
142. **}**
143. **}**
144. **}**
145. **private** void MoveToRandomReachable**(**Unit unit**,** bool AwayFromHQ **=** **false,** int attempts **=** 10**)**
146. **{**
147. reachableTiles **=** GetReachableTiles**(**unit**);**
148. int randomTile**;**
149. **if** **(**AwayFromHQ**)**
150. **{**
151. Tile previousTile**;**
152. Tile newTile **=** **null;**
153. Tile destination **=** **null;**
154. **for** **(**int i **=** 0**;** i **<** attempts**;** i**++)**
155. **{**
156. randomTile **=** random**.**Next**(**reachableTiles**.**Count**);**
157. **if** **(**newTile **==** **null)**
158. **{**
159. previousTile **=** reachableTiles**[**randomTile**];**
160. **}**
161. **else**
162. **{**
163. previousTile **=** newTile**;**
164. **}**
165. newTile **=** reachableTiles**[**randomTile**];**
166. **if** **(**GetSquaredDistance**(**newTile**,** HQ**)** **>=** GetSquaredDistance**(**previousTile**,** HQ**))**
167. **{**
168. destination **=** newTile**;**
169. **}**
170. **}**
171. **if** **(**newTile **!=** **null)**
172. **{**
173. unit**.**Fuel**--;**
174. unit**.**State **=** UnitState**.**Moved**;**
175. unit**.**Position **=** destination**.**Position**;**
176. **}**
177. **}**
178. **else**
179. **{**
180. unit**.**Fuel**--;**
181. unit**.**State **=** UnitState**.**Moved**;**
182. randomTile **=** random**.**Next**(**reachableTiles**.**Count**);**
183. unit**.**Position **=** reachableTiles**[**randomTile**].**Position**;**
184. **}**
185. **}**
186. **private** List**<**Tile**>** GetReachableTiles**(**Unit unit**)**
187. **{**
188. **return** \_gameUI**.**GetReachableTiles**(**unit**,** \_unitManager**.**GetUnitPositions**(),** \_inputController**.**GetUnitTile**(**unit**));**
189. **}**
190. **private** bool EnemyCloseToHQ**()**
191. **{**
192. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
193. **{**
194. reachableTiles **=** GetReachableTiles**(**unit**);**
195. **if** **(**unit**.**Team **!=** Team **&&** reachableTiles**.**Contains**(**HQ**))**
196. **{**
197. **return** **true;**
198. **}**
199. **}**
200. **return** **false;**
201. **}**
202. **private** bool Winning**()**
203. **{**
204. **foreach** **(**Tile EnemyHQ **in** EnemyHQs**)**
205. **{**
206. int numOfUnits **=** 0**;**
207. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
208. **{**
209. **if** **(**unit**.**Team **==** EnemyHQ**.**Team**)**
210. **{**
211. numOfUnits**++;**
212. **}**
213. **}**
214. **if** **(**AIUnits**.**Count **>** numOfUnits**)**
215. **{**
216. HQToAttack **=** EnemyHQ**;**
217. **return** **true;**
218. **}**
219. **}**
220. **return** **false;**
221. **}**
222. **private** bool CloseByStructure**()**
223. **{**
224. closeByStructures**.**Clear**();**
225. **foreach** **(**Unit unit **in** AIUnits**)**
226. **{**
227. reachableTiles **=** GetReachableTiles**(**unit**);**
228. **foreach** **(**Tile tile **in** reachableTiles**)**
229. **{**
230. **if** **((**tile**.**Type **==** TileType**.**City
231. **||** tile**.**Type **==** TileType**.**Factory**)**
232. **&&** tile**.**Team **!=** Team**)**
233. **{**
234. closeByStructures**.**Add**(**tile**);**
235. **}**
236. **}**
237. **}**
238. **if** **(**closeByStructures**.**Count **>** 0**)**
239. **{**
240. **return** **true;**
241. **}**
242. **return** **false;**
243. **}**
244. **private** void AttackNeighbouringUnits**()**
245. **{**
246. **foreach** **(**Unit unit **in** AIUnits**)**
247. **{**
248. Tile startingTile **=** \_inputController**.**GetUnitTile**(**unit**);**
249. List**<**Tile**>** adjacentTiles **=** \_gameMap**.**GetNeighbours**(**startingTile**);**
250. **foreach** **(**Tile neighbour **in** adjacentTiles**)**
251. **{**
252. Unit neighbouringUnit **=** \_inputController**.**GetTileUnit**(**neighbour**);**
253. **if** **(**neighbouringUnit **!=** **null**
254. **&&** neighbouringUnit**.**Team **!=** Team
255. **&&** unit**.**Ammo **>** 0
256. **)**
257. **{**
258. unit**.**Health **-=** \_gameUI**.**CalculateDamage**(**unit**,** neighbouringUnit**);**
259. unit**.**Ammo**--;**
260. **if** **(**neighbouringUnit**.**Health **>** 0 **&&** neighbouringUnit**.**Ammo **>** 0**)**
261. **{**
262. neighbouringUnit**.**Health **-=** \_gameUI**.**CalculateDamage**(**neighbouringUnit**,** unit**);**
263. neighbouringUnit**.**Ammo**--;**
264. **}**
265. unit**.**State **=** UnitState**.**Used**;**
266. **break;**
267. **}**
268. **}**
269. **}**
270. **}**
271. **private** void MoveTowardsTile**(**Unit movingUnit**,** Tile destination**)**
272. **{**
273. reachableTiles**.**Clear**();**
274. reachableTiles **=** \_gameUI**.**GetReachableTiles**(**movingUnit**,** \_unitManager**.**GetUnitPositions**(),** \_inputController**.**GetUnitTile**(**movingUnit**));**
275. **if** **(**movingUnit**.**Position **!=** destination**.**Position**)**
276. **{**
277. Unit destinationUnit **=** \_inputController**.**GetTileUnit**(**destination**);**
278. **if** **(**destination**.**Team **!=** Team **||** destinationUnit **!=** **null** **&&** destinationUnit**.**Team **!=** Team**)**//HERE
279. **{**
280. Tile closestTile **=** **null;**
281. float dist **=** int**.**MaxValue**;**
282. **foreach** **(**Tile tile **in** reachableTiles**)**
283. **{**
284. **if** **(**GetSquaredDistance**(**tile**,** destination**)** **<** dist**)**
285. **{**
286. closestTile **=** tile**;**
287. dist **=** GetSquaredDistance**(**tile**,** destination**);**
288. **}**
289. **}**
290. **if** **(**closestTile **!=** **null)**
291. **{**
292. movingUnit**.**Position **=** closestTile**.**Position**;**
293. movingUnit**.**State **=** UnitState**.**Moved**;**
294. movingUnit**.**Fuel**--;**
295. **}**
296. **}**
297. **}**
298. **}**
299. **private** static float GetSquaredDistance**(**Tile startingTile**,** Tile destinationTile**)**
300. **{**
301. float x **=** startingTile**.**Position**.**X **-** destinationTile**.**Position**.**X**;**
302. float y **=** startingTile**.**Position**.**Y **-** destinationTile**.**Position**.**Y**;**
303. **return** x **\*** x **+** y **\*** y**;**
304. **}**
305. **private** void GetEnemyHQs**()**
306. **{**
307. EnemyHQs**.**Clear**();**
308. **foreach** **(**Tile structure **in** \_gameMap**.**HQs**)**
309. **{**
310. **if** **(**structure**.**Team **!=** Team**)**
311. **{**
312. EnemyHQs**.**Add**(**structure**);**
313. **}**
314. **else**
315. **{**
316. HQ **=** structure**;**
317. **}**
318. **}**
319. **}**
320. **private** void GetFriendlyBuildings**()**
321. **{**
322. FriendlyBuildings**.**Clear**();**
323. **foreach** **(**Tile structure **in** \_gameMap**.**Structures**)**
324. **{**
325. **if** **(**structure**.**Team **==** Team**)**
326. **{**
327. FriendlyBuildings**.**Add**(**structure**);**
328. **}**
329. **}**
330. **}**
331. **private** void GetAIUnits**()**
332. **{**
333. AIUnits**.**Clear**();**
334. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
335. **{**
336. **if** **(**unit**.**Team **==** Team**)**
337. **{**
338. AIUnits**.**Add**(**unit**);**
339. **}**
340. **}**
341. **}**
342. **private** void ProduceUnits**()**
343. **{**
344. GetFriendlyBuildings**();**
345. **foreach** **(**Unit unit **in** AIUnits**)**
346. **{**
347. **if** **(**unit**.**State **!=** UnitState**.**Moved
348. **&&** unit**.**State **!=** UnitState**.**Used
349. **&&** unit**.**Fuel **>** 0
350. **)**
351. **{**
352. MoveToRandomReachable**(**unit**,** **true);**
353. **}**
354. **}**
355. **foreach** **(**Tile structure **in** FriendlyBuildings**)**
356. **{**
357. **if** **((**structure**.**Type **==** TileType**.**Factory
358. **||** structure**.**Type **==** TileType**.**HQ**)**
359. **&&** \_inputController**.**GetTileUnit**(**structure**)** **==** **null**
360. **)**
361. **{**
362. int unitType **=** **-**1**;**
363. **if** **(**Funds **>=** 7000**)**
364. **{**
365. Funds **-=** 7000**;**
366. unitType **=** 3**;**
367. **}**
368. **else** **if** **(**Funds **>=** 5000**)**
369. **{**
370. **continue;**
371. **}**
372. **else** **if** **(**Funds **>=** 4000**)**
373. **{**
374. Funds **-=** 3000**;**
375. unitType **=** 2**;**
376. **}**
377. **else**
378. **{**
379. Funds **-=** 1000**;**
380. unitType **=** 1**;**
381. **}**
382. **if** **(**unitType **!=** **-**1**)**
383. **{**
384. Unit newUnit **=** **new(**Texture**,** structure**.**Position**,** unitType**,** Team**)**
385. **{**
386. State **=** UnitState**.**Used
387. **};**
388. \_unitManager**.**AddUnit**(**newUnit**);**
389. **}**
390. **}**
391. **}**
392. **}**
393. **}**
394. **}**

#### InputController.cs

1. **using** Basic\_Wars\_V2**.**Entities**;**
2. **using** Microsoft**.**Xna**.**Framework**;**
3. **using** Microsoft**.**Xna**.**Framework**.**Input**;**
4. **namespace** Basic\_Wars\_V2**.**System
5. **{**
6. **public** class InputController
7. **{**
8. **private** UnitManager \_unitManager**;**
9. **private** **readonly** ButtonManager \_buttonManager**;**
10. **private** MapManager \_gameMap**;**
11. **private** MouseState currentMouseState**;**
12. **private** MouseState previousMouseState**;**
13. **public** Rectangle MouseCollider **{** **get;** **private** **set;** **}**
14. **public** InputController**(**UnitManager unitManager**,** ButtonManager buttonManager**,** MapManager gameMap**)**
15. **{**
16. \_unitManager **=** unitManager**;**
17. \_buttonManager **=** buttonManager**;**
18. \_gameMap **=** gameMap**;**
19. **}**
20. **public** void UpdateMouseState**()**
21. **{**
22. previousMouseState **=** currentMouseState**;**
23. currentMouseState **=** Mouse**.**GetState**();**
24. MouseCollider **=** **new** Rectangle**(**currentMouseState**.**X**,** currentMouseState**.**Y**,** 1**,** 1**);**
25. **}**
26. **public** bool LeftMouseClicked**()**
27. **{**
28. **if** **(**previousMouseState**.**LeftButton **==** ButtonState**.**Released
29. **&&** currentMouseState**.**LeftButton **==** ButtonState**.**Pressed
30. **)**
31. **{**
32. **return** **true;**
33. **}**
34. **return** **false;**
35. **}**
36. **public** void ProcessControls**(**GameTime gameTime**,** bool ProcessButtonsOnly**)**
37. **{**
38. bool UnitSelected **=** **false;**
39. UpdateMouseState**();**
40. **if** **(!**ProcessButtonsOnly**)**
41. **{**
42. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
43. **{**
44. **if** **(**
45. MouseCollider**.**Intersects**(**unit**.**Collider**)**
46. **&&** LeftMouseClicked**()**
47. **)**
48. **{**
49. UnitSelected **=** **true;**
50. unit**.**Selected **=** **true;**
51. **}**
52. **}**
53. **if** **(!**UnitSelected**)**
54. **{**
55. **foreach** **(**Tile tile **in** \_gameMap**.**Map**)**
56. **{**
57. **if** **(**
58. MouseCollider**.**Intersects**(**tile**.**Collider**)**
59. **&&** LeftMouseClicked**()**
60. **)**
61. **{**
62. tile**.**Selected **=** **true;**
63. **}**
64. **}**
65. **}**
66. **}**
67. **foreach** **(**Button button **in** \_buttonManager**.**buttons**)**
68. **{**
69. **if** **(**
70. MouseCollider**.**Intersects**(**button**.**Collider**)**
71. **&&** LeftMouseClicked**()**
72. **)**
73. **{**
74. button**.**Pressed **=** **true;**
75. **}**
76. **}**
77. **}**
78. **public** void ChangeMap**(**MapManager newMap**)**
79. **{**
80. \_gameMap **=** newMap**;**
81. **}**
82. **public** Unit GetSelectedUnit**()**
83. **{**
84. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
85. **{**
86. **if** **(**unit**.**Selected**)**
87. **{**
88. **return** unit**;**
89. **}**
90. **}**
91. **return** **null;**
92. **}**
93. **public** Tile GetSelectedTile**()**
94. **{**
95. **foreach** **(**Tile tile **in** \_gameMap**.**Map**)**
96. **{**
97. **if** **(**tile**.**Selected**)**
98. **{**
99. tile**.**Selected **=** **false;**
100. **return** tile**;**
101. **}**
102. **}**
103. **return** **null;**
104. **}**
105. **public** Button GetButtonPressed**()**
106. **{**
107. **foreach** **(**Button button **in** \_buttonManager**.**buttons**)**
108. **{**
109. **if** **(**button**.**Pressed**)**
110. **{**
111. button**.**Pressed **=** **false;**
112. **return** button**;**
113. **}**
114. **}**
115. **return** **null;**
116. **}**
117. **public** Tile GetUnitTile**(**Unit unit**)**
118. **{**
119. **foreach** **(**Tile tile **in** \_gameMap**.**Map**)**
120. **{**
121. **if** **(**unit**.**Position **==** tile**.**Position**)**
122. **{**
123. **return** tile**;**
124. **}**
125. **}**
126. **return** **null;**
127. **}**
128. **public** Unit GetTileUnit**(**Tile tile**)**
129. **{**
130. **foreach** **(**Unit unit **in** \_unitManager**.**units**)**
131. **{**
132. **if** **(**tile**.**Position **==** unit**.**Position**)**
133. **{**
134. **return** unit**;**
135. **}**
136. **}**
137. **return** **null;**
138. **}**
139. **}**
140. **}**

#### Dijkstra.cs

1. **using** Basic\_Wars\_V2**.**Entities**;**
2. **using** System**.**Collections**.**Generic**;**
3. **namespace** Basic\_Wars\_V2**.**System
4. **{**
5. **public** class Dijkstra
6. **{**
7. **private** **readonly** MapManager mapManager**;**
8. **public** Dijkstra**(**MapManager Map**)**
9. **{**
10. mapManager **=** Map**;**
11. **}**
12. **public** List**<**Tile**>** FindReachableTiles**(**Tile startingTile**,** Unit unit**)**
13. **{**
14. List**<**Tile**>** reachableTiles **=** **new();**
15. PriorityQueue**<**Tile**,** int**>** queue **=** **new();**
16. startingTile**.**TotalCost **=** 0**;**
17. queue**.**Enqueue**(**startingTile**,** 0**);**
18. **while** **(**queue**.**Count **>** 0**)**
19. **{**
20. Tile currentTile **=** queue**.**Dequeue**();**
21. reachableTiles**.**Add**(**currentTile**);**
22. **foreach** **(**Tile neighbor **in** mapManager**.**GetNeighbours**(**currentTile**))**
23. **{**
24. int cost **=** currentTile**.**TotalCost **+** mapManager**.**GetCost**(**currentTile**,** unit**);**
25. **if** **(!**reachableTiles**.**Contains**(**neighbor**)** **&&** cost **<=** unit**.**MovementPoints**)**
26. **{**
27. neighbor**.**TotalCost **=** cost**;**
28. queue**.**Enqueue**(**neighbor**,** cost**);**
29. **}**
30. **}**
31. **}**
32. **return** reachableTiles**;**
33. **}**
34. **}**
35. **}**

#### PoissonDiscSampling.cs

1. **using** Microsoft**.**Xna**.**Framework**;**
2. **using** System**;**
3. **using** System**.**Collections**.**Generic**;**
4. **namespace** Basic\_Wars\_V2**.**System
5. **{**
6. **public** static class PoissonDiscSampling
7. **{**
8. **public** static List**<**Vector2**>** GetPoints**(**double radius**,** Vector2 mapSize**,** int k **=** 30**)**
9. **{**
10. Random random **=** **new();**
11. double cellSize **=** radius **/** Math**.**Sqrt**(**2**);**
12. int Width **=** **(**int**)(**mapSize**.**X **/** cellSize**)** **+** 1**;**
13. int Height **=** **(**int**)(**mapSize**.**Y **/** cellSize**)** **+** 1**;**
14. int**[,]** grid **=** **new** int**[**Width**,** Height**];**
15. List**<**Vector2**>** points **=** **new();**
16. List**<**Vector2**>** startPoints **=** **new();**
17. Vector2 initialPoint **=** **new(**random**.**Next**((**int**)**mapSize**.**X**),** random**.**Next**((**int**)**mapSize**.**Y**));**
18. startPoints**.**Add**(**initialPoint**);**
19. **while** **(**startPoints**.**Count **>** 0**)**
20. **{**
21. int randomIndex **=** random**.**Next**(**startPoints**.**Count**);**
22. Vector2 startPoint **=** startPoints**[**randomIndex**];**
23. bool potentialPointFound **=** **false;**
24. **for** **(**int tries **=** 0**;** tries **<** k**;** tries**++)**
25. **{**
26. double angle **=** random**.**NextDouble**()** **\*** 2 **\*** Math**.**PI**;**
27. Vector2 direction **=** **new((**float**)**Math**.**Sin**(**angle**),** **(**float**)**Math**.**Cos**(**angle**));**
28. Vector2 candidatePoint **=** startPoint **+** direction **\*** random**.**Next**((**int**)**radius**,** **(**int**)**radius **\*** 2**);**
29. **if** **(**IsValid**(**candidatePoint**,** mapSize**,** cellSize**,** points**,** grid**,** radius**))**
30. **{**
31. points**.**Add**(**candidatePoint**);**
32. startPoints**.**Add**(**candidatePoint**);**
33. grid**[(**int**)(**candidatePoint**.**X **/** cellSize**),** **(**int**)(**candidatePoint**.**Y **/** cellSize**)]** **=** points**.**Count**;**
34. potentialPointFound **=** **true;**
35. **break;**
36. **}**
37. **}**
38. **if** **(!**potentialPointFound**)**
39. **{**
40. startPoints**.**RemoveAt**(**randomIndex**);**
41. **break;**
42. **}**
43. **}**
44. **return** points**;**
45. **}**
46. **private** static bool IsValid**(**Vector2 potentialPoint**,** Vector2 mapSize**,** double cellSize**,** List**<**Vector2**>** points**,** int**[,]** grid**,** double radius**)**
47. **{**
48. **if** **(**potentialPoint**.**X **>=** 0 **&&** potentialPoint**.**X **<** mapSize**.**X **&&** potentialPoint**.**Y **>=** 0 **&&** potentialPoint**.**Y **<** mapSize**.**Y**)**
49. **{**
50. Vector2 cell **=** **new((**float**)(**potentialPoint**.**X **/** cellSize**),** **(**float**)(**potentialPoint**.**Y **/** cellSize**));**
51. int startX **=** Math**.**Max**(**0**,** **(**int**)**cell**.**X **-** 2**);**
52. int endX **=** Math**.**Min**((**int**)**cell**.**X **+** 2**,** grid**.**GetLength**(**0**)** **-** 1**);**
53. int startY **=** Math**.**Max**(**0**,** **(**int**)**cell**.**Y **-** 2**);**
54. int endY **=** Math**.**Min**((**int**)**cell**.**Y **+** 2**,** grid**.**GetLength**(**1**)** **-** 1**);**
55. **for** **(**int x **=** startX**;** x **<=** endX**;** x**++)**
56. **{**
57. **for** **(**int y **=** startY**;** y **<=** endY**;** y**++)**
58. **{**
59. int pointIndex **=** grid**[**x**,** y**]** **-** 1**;**
60. **if** **(**pointIndex **!=** **-**1**)**
61. **{**
62. double squrdDistance **=** **(**SquaredDistance**(**potentialPoint**,** points**[**pointIndex**]));**
63. **if** **(**squrdDistance **<** radius **\*** radius**)**
64. **{**
65. **return** **false;**
66. **}**
67. **}**
68. **}**
69. **}**
70. **return** **true;**
71. **}**
72. **return** **false;**
73. **}**
74. **private** static double SquaredDistance**(**Vector2 start**,** Vector2 end**)**
75. **{**
76. double x **=** start**.**X **-** end**.**X**;**
77. double y **=** start**.**Y **-** end**.**Y**;**
78. **return** x **\*** x **+** y **\*** y**;**
79. **}**
80. **}**
81. **}**

#### GameData.cs

1. **using** System**;**
2. **using** System**.**Collections**.**Generic**;**
3. **namespace** Basic\_Wars\_V2**.**System
4. **{**
5. **[**Serializable**]**
6. **public** class GameData
7. **{**
8. **public** int MapWidth **{** **get;** **set;** **}**
9. **public** int MapHeight **{** **get;** **set;** **}**
10. **public** List**<**UnitData**>** Units **{** **get;** **set;** **}**
11. **public** List**<**TileData**>** Map **{** **get;** **set;** **}**
12. **public** List**<**TileData**>** Structures **{** **get;** **set;** **}**
13. **public** List**<**PlayerData**>** Players **{** **get;** **set;** **}**
14. **public** GameStateData GameStateData **{** **get;** **set;** **}**
15. **public** AIData Computer **{** **get;** **set;** **}**
16. **public** GameData**()** **{** **}**
17. **public** GameData**(**List**<**UnitData**>** units**,** List**<**TileData**>** gameMap**,** List**<**TileData**>** structures**,** List**<**PlayerData**>** players**,** GameStateData gameStateData**,** int mapWidth**,** int mapHeight**,** AIData computer **=** **null)**
18. **{**
19. Units **=** units**;**
20. Map **=** gameMap**;**
21. Structures **=** structures**;**
22. Players **=** players**;**
23. Computer **=** computer**;**
24. GameStateData **=** gameStateData**;**
25. MapWidth **=** mapWidth**;**
26. MapHeight **=** mapHeight**;**
27. **}**
28. **}**
29. **}**

#### GameStateData.cs

1. **using** System**;**
2. **namespace** Basic\_Wars\_V2**.**System
3. **{**
4. **[**Serializable**]**
5. **public** class GameStateData
6. **{**
7. **public** int TurnNumber **{** **get;** **set;** **}**
8. **public** int CurrentPlayerIndex **{** **get;** **set;** **}**
9. **public** bool AddAI **{** **get;** **set;** **}**
10. **public** GameStateData**()** **{** **}**
11. **public** GameStateData**(**int turnNumber**,** int currentPlayerIndex**,** bool addAI**)**
12. **{**
13. TurnNumber **=** turnNumber**;**
14. CurrentPlayerIndex **=** currentPlayerIndex**;**
15. AddAI **=** addAI**;**
16. **}**
17. **}**
18. **}**

#### PlayerData.cs

1. **using** Basic\_Wars\_V2**.**Entities**;**
2. **using** System**;**
3. **namespace** Basic\_Wars\_V2**.**System
4. **{**
5. **[**Serializable**]**
6. **public** class PlayerData
7. **{**
8. **public** int Team **{** **get;** **set;** **}**
9. **public** int Funds **{** **get;** **set;** **}**
10. **public** bool HasHQ **{** **get;** **set;** **}**
11. **public** string Colour **{** **get;** **set;** **}**
12. **public** bool IsAI **{** **get;** **set;** **}**
13. **public** PlayerData**()** **{** **}**
14. **public** PlayerData**(**Player player**)**
15. **{**
16. Team **=** player**.**Team**;**
17. Funds **=** player**.**Funds**;**
18. HasHQ **=** player**.**HasHQ**;**
19. Colour **=** player**.**Colour**;**
20. IsAI **=** player**.**IsAI**;**
21. **}**
22. **public** Player FromPlayerData**()**
23. **{**
24. Player player **=** **new** Player**(**Team**,** Funds**)**
25. **{**
26. HasHQ **=** HasHQ**,**
27. Colour **=** Colour**,**
28. IsAI **=** IsAI
29. **};**
30. **return** player**;**
31. **}**
32. **}**
33. **}**

#### AIData.cs

1. **using** Basic\_Wars\_V2**.**Entities**;**
2. **using** Basic\_Wars\_V2**.**Enums**;**
3. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
4. **using** System**;**
5. **namespace** Basic\_Wars\_V2**.**System
6. **{**
7. **[**Serializable**]**
8. **public** class AIData **:** PlayerData
9. **{**
10. **public** AIState AIState **{** **get;** **set;** **}**
11. **public** AIData**()** **{** **}**
12. **public** AIData**(**AI Computer**)**
13. **{**
14. Team **=** Computer**.**Team**;**
15. Funds **=** Computer**.**Funds**;**
16. HasHQ **=** Computer**.**HasHQ**;**
17. Colour **=** Computer**.**Colour**;**
18. **}**
19. **public** AI FromAIData**(**Texture2D Texture**,** MapManager mapManager**,** UnitManager unitManager**,** GameUI gameUI**,** InputController inputController**)**
20. **{**
21. AI Computer **=** **new(**Team**,** Funds**,** mapManager**,** unitManager**,** gameUI**,** inputController**,** Texture**)**
22. **{**
23. HasHQ **=** HasHQ**,**
24. Colour **=** Colour
25. **};**
26. **return** Computer**;**
27. **}**
28. **}**
29. **}**

#### UnitData.cs

1. **using** Basic\_Wars\_V2**.**Entities**;**
2. **using** Basic\_Wars\_V2**.**Enums**;**
3. **using** Microsoft**.**Xna**.**Framework**;**
4. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
5. **using** System**;**
6. **namespace** Basic\_Wars\_V2**.**System
7. **{**
8. **[**Serializable**]**
9. **public** class UnitData
10. **{**
11. **public** Vector2 Position **{** **get;** **set;** **}**
12. **public** UnitState State **{** **get;** **set;** **}**
13. **public** int Team **{** **get;** **set;** **}**
14. **public** int UnitTypeInt **{** **get;** **set;** **}**
15. **public** int Health **{** **get;** **set;** **}**
16. **public** int Ammo **{** **get;** **set;** **}**
17. **public** int Fuel **{** **get;** **set;** **}**
18. **public** float Defence **{** **get;** **set;** **}**
19. **public** int MovementPoints **{** **get;** **set;** **}**
20. **public** int CostToProduce **{** **get;** **set;** **}**
21. **public** UnitData**()** **{** **}**
22. **public** UnitData**(**Unit unit**)**
23. **{**
24. Position **=** unit**.**Position**;**
25. State **=** unit**.**State**;**
26. Team **=** unit**.**Team**;**
27. UnitTypeInt **=** unit**.**UnitTypeInt**;**
28. Health **=** unit**.**Health**;**
29. Ammo **=** unit**.**Ammo**;**
30. Fuel **=** unit**.**Fuel**;**
31. MovementPoints **=** unit**.**MovementPoints**;**
32. CostToProduce **=** unit**.**CostToProduce**;**
33. **}**
34. **public** Unit FromUnitData**(**Texture2D Texture**)**
35. **{**
36. Unit unit **=** **new(**Texture**,** Position**,** UnitTypeInt**,** Team**)**
37. **{**
38. Health **=** Health**,**
39. Ammo **=** Ammo**,**
40. Fuel **=** Fuel**,**
41. Defence **=** Defence**,**
42. MovementPoints **=** MovementPoints**,**
43. CostToProduce **=** CostToProduce
44. **};**
45. unit**.**CreateUnitSprite**(**UnitTypeInt**);**
46. **return** unit**;**
47. **}**
48. **}**
49. **}**

#### TileData.cs

1. **using** Basic\_Wars\_V2**.**Entities**;**
2. **using** Basic\_Wars\_V2**.**Enums**;**
3. **using** Microsoft**.**Xna**.**Framework**;**
4. **using** Microsoft**.**Xna**.**Framework**.**Graphics**;**
5. **using** System**;**
6. **namespace** Basic\_Wars\_V2**.**System
7. **{**
8. **[**Serializable**]**
9. **public** class TileData
10. **{**
11. **public** TileType Type **{** **get;** **set;** **}**
12. **public** Vector2 Position **{** **get;** **set;** **}**
13. **public** Vector2 MapGridPos **{** **get;** **set;** **}**
14. **public** int Team **{** **get;** **set;** **}**
15. **public** TileData**()** **{** **}**
16. **public** TileData**(**Tile tile**)**
17. **{**
18. Type **=** tile**.**Type**;**
19. Position **=** tile**.**Position**;**
20. MapGridPos **=** tile**.**MapGridPos**;**
21. Team **=** tile**.**Team**;**
22. **}**
23. **public** Tile FromTileData**(**Texture2D Texture**)**
24. **{**
25. Tile tile **=** **new** Tile**(**Position**,** Texture**,** Team**)**
26. **{**
27. Type **=** Type**,**
28. MapGridPos **=** MapGridPos
29. **};**
30. **return** tile**;**
31. **}**
32. **}**
33. **}**

# Testing

## Game Menus

Testing of the game’s menus with evidence in video format:

<https://youtu.be/vEUIU8kvp_U>

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Test Data | Expected Outcome | Test Outcome |
| New Game | Button Press | User is taken to New Game Menu | Passed |
| Load Game  (No GameData File) | Button Press | Nothing occurs | Passed |
| Quit Game | Button Press | Application exits | Passed |
| Menu Button (New Game Menu) | Button Press | Returns the user to the Main Menu | Passed |
| Refresh Map | Button Press | The map is regenerated and displayed | Passed |
| Player 2, 3, 4 | Button Press | The number of players and HQs should change according to the button selected | Passed |
| Save | Button Press | Game is saved into an XML file located in in the same directory as the game file | Passed |
| Load | Button Press | XML file is loaded back into the game for the player to continue playing | Passed |
| Pause | Button Press | The game should pause, and inputs should be disabled to everything except the paused menu | Passed |

## Unit Movement

Testing of unit movement with evidence in video format:

<https://youtu.be/RUjwbfvxylk>

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Test Data | Expected Outcome | Test Outcome |
| Vehicle allowed movement | APC unit | Unit should not be able to move onto mountains | Passed |
| Vehicle movement on roads | APC unit | Unit should be able to move much further on roads | Passed |
| Foot units’ movement on mountains | Infantry unit | Unit range should be reduced while on a mountain | Passed |
| Units moving onto other units | Infantry units | Units should not be able to move onto a tile where another unit is present | Passed |
| Units moving only on tiles | Infantry units | Units should only be able to move on map tiles | Passed |

## Unit Capture

For the sake of speeding up testing the unit can move continuously during the same turn.

Testing of unit capture ability with evidence in video format:

<https://youtu.be/81q3Ti8IGSI>

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Test Data | Expected Outcome | Test Outcome |
| Unit vehicles capture | APC unit | Vehicle units should not be able to capture structures | Passed |
| Unit infantry capture | Infantry unit | The unit should be able to capture structures | Passed |
| Unit capture friendly structures | Infantry unit | The capture option should not be displayed when the unit is located on a friendly structure | Passed |
| Unit capture enemy structures | Infantry unit | The unit should be able to capture enemy structures | Passed |

## Unit Resupply

Testing of unit resupply ability with evidence in video format:

<https://youtu.be/dankUEr0jq8>

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Test Data | Expected Outcome | Test Outcome |
| Units resupply at friendly structure | Infantry unit | Units should be able to resupply on a structure if it is from its own team | Passed |
| Units resupply at non friendly structure | Infantry unit | Units should not be able to resupply from a structure not from its own team | Failed – Although the resupply option is not displayed, it can still be selected as an option this is likely a small error in an IF statement |
| APC resupply units | APC unit, Infantry unit | APC should be able to resupply adjacent to it | Passed |
| APC resupply itself | APC unit | APC should not be able to resupply itself unless it is also on a friendly structure | Failed – See Test 2 |

## Unit Attack

Testing of unit resupply ability with evidence in video format:

<https://youtu.be/JCG6OguS9iA>

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Test Data | Expected Outcome | Test Outcome |
| Units attack | Infantry unit | Units should not be able to attack if there are no adjacent enemies | Passed |
| Unit attack friendly | Infantry unit | Units should not be able to attack friendly units | Passed |
| Unit attack enemy | Infantry unit | Units should be able to attack and deal damage to an enemy | Passed |
| Unit counterattack | Infantry unit | Units should receive damage from a counterattack | Passed |
| Unit dead | Infantry unit | Units with less than 0 health should be removed from the game | Passed |

# Evaluation

## Evaluation of Objectives

Whether an objective has been met, partially met, or not met is indicated by colours.

|  |
| --- |
| Objective status |
| Met |
| Partially Met |
| Not Met |

*Below each objective will be an explanation behind the reasoning of the status if necessary*

### 1.6.1. Menus

1.6.1.1. The main menu of the game is clear and easy to understand with defined options on proceeding to the game

*Main is simple in design and contains the 3 options for proceeding to the game*

1.6.1.2. The main menu should display 3 options: New game, Load game and Quit

1.6.1.3. Within the game the GUI should clearly display all the information the player will need

Current Player

Current balance

Turn number

End turn button

*While playing the game the current balance/current player and turn number are displayed in the top left and top right corners respectively and the end turn button is located at the bottom left corner of the window*

1.6.1.4. In-game menu should display 3 options: Resume, Save game, Quit game

*When the game is paused the 3 options of Resume, Save and Quit are presented to the user*

### 1.6.2. New Game

1.6.2.1. Game options for user to configure

The player can continue randomising the map

Number of players (2 – 4)

Whether a player should be an AI

*Not all players are able to be AI, only one of the players can be an AI*

1.6.2.2. All players will start with a balance of 1000

### 1.6.3. Game Map

1.6.3.1. The map will be randomly generated

*Map is randomly generated within the MapManager class*

1.6.3.2. Structures on the map should be distributed around the map and not focused in a particular area

*The algorithm for generating structures on the map is quite inconsistent and only sometimes works in the way intended*

1.6.3.3. Roads should run between some of the structures in the map

*Roads are generated between structures on the map. However, there are corners for the roads making them look slightly out of place at times*

1.6.3.4. HQs on the map should be spread as far apart as possible

*The HQs are placed in opposite corners of the map to be as far apart as possible*

1.6.3.5. All buildings will initially be neutral other than player HQs

*When generating buildings, the Team attribute is set to neutral and building colours are also neutral*

### 1.6.4. Save/Load game

1.6.4.1. Game will be saved into an XML or JSON file

*Game is saved into an XML file*

1.6.4.2. Saves:

Map

Structures on the map and their statistics (e.g team, defence bonus)

Game state

All unit positions and statistics (e.g health, team, remaining actions etc)

Which player’s turn it is

Whether a player is an AI

Turn number

*Game saves all attributes listed above*

1.6.4.3. Saved data is all loaded so that the game can be continued from where it was left

*Saved data can be loaded and the player is able to start at the point they left off. However, there seems to be a bug that I haven’t been able to fix where units are able to perform further actions when they shouldn’t*

### 1.6.5. Players

1.6.5.1. Will have a Boolean value determining whether they are an AI

1.6.5.2. Start the game with an HQ tile where units can be produced

*Players start the game with 1000 funds so that they can produce a unit on their HQ*

1.6.5.3. Players that no longer have an HQ (due to capture) are taken out of the game unless they own another HQ

*Players are removed from the game once they lose their HQ*

1.6.5.4. All players in the game must be contained within a list to loop through whenever a player ends their turn

1.6.5.5. All players should have a unique colour to identify them and any buildings they own

Players have unique colour to identify the team they are on: Red, Blue, Green, and Yellow

### 1.6.6. Artificial Intelligence (AI)

1.6.6.1. The AI will be able to attack, manoeuvre and create units as well as capturing buildings

*The AI can do all the actions listed above*

1.6.6.1.1. The AI should try to attack the player when it is in an advantageous position

*The AI is not entirely successful in this objective since it’s definition of an advantageous position is when it has more units and is not based on other factors such as: enemy unit positions, unit type, funds, or unit health*

1.6.6.1.2. The AI should attack units that are in range of its own units

1.6.6.1.3. If a building can be captured, the AI should attempt to capture it

1.6.6.1.4. The AI should attempt to create units at all possible production locations

### 1.6.7. Attacking

1.6.7.1. When attacking, the user should be able to view the statistics of both the friendly attacking unit and the enemy defending unit

*The user is not able to view the enemy unit’s statistics once the attacking state has been selected. However, the player can do this before selecting the attack option*

1.6.7.2. The defending unit will lose health from taking damage

1.6.7.3. After attacking the attacking unit will take damage from the defending unit as a counterattack

1.6.7.4. After attacking, the unit will become immobile till the start of a new turn

1.6.7.5. To indicate which unit the player is attacking, the enemy unit should be highlighted

### 1.6.8. Tiles

1.6.8.1. Different tiles will have different attributes

1.6.8.2. Roads

Improved mobility for vehicles

Units will have no defence bonus

1.6.8.3. Plains

Mobility of vehicles is lowered

1.6.8.4. Mountain

Units on a mountain will have a greater defence bonus

Non vehicles can traverse mountains

Moving onto a mountain reduces mobility

1.6.8.5. Buildings – can be captured, produces 1000 funds per turn and can resupply units

1.6.8.6. HQ

Can produce units

1.6.8.7. Cities

Units will take reduced damage while on this tile

1.6.8.8. Factories

Produces units

### 1.6.9. Selecting/Manoeuvring units

1.6.9.1. Units can only be moved once per turn

1.6.9.2. Only one unit is able to be selected at any one time

1.6.9.3. Units are unable to move onto a tile where a unit is already present

1.6.9.4. Newly created units are unable to move for a turn

1.6.9.5. Units have a limited amount of fuel and must be resupplied when they have run out to move again

1.6.9.6. The unit selected shows its stats in a small list

1.6.9.7. When moving a selected unit, the possible tiles that it can move to should be highlighted to the player

### 1.6.10. Units

1.6.10.1. Different units will have different attributes

1.6.10.2. A unit’s team should be identifiable by the colour of the unit

1.6.10.3. Infantry

Basic starting unit with well-rounded stats

Can capture buildings

1.6.10.4. Mech

Greater damage against vehicles

Can capture buildings

1.6.10.5. APC

Ability to resupply adjacent units

Ability to carry other units that are not vehicles

Cannot capture buildings

*The implementation for the APC to carry nonvehicle units was not accomplished. However, it can resupply adjacent units*

1.6.10.6. Tank

High Damage

High armour

Low mobility and fuel

Expensive

Cannot capture buildings

1.6.10.7. All units will have:

Health

Ammo

Fuel

Defence

Mobility distance

### 1.6.11. Game

1.6.11.1. Win condition of game is when there is only 1 remaining HQ left

*The game checks the winning condition after a capture has occurred to reduce the number of calls to check whether the game is over.*

1.6.11.2. Units can be produced on HQ and factory tiles given there is not already a unit there

*When a unit is present on a tile that can produce units, the tile is unable to be selected until the unit has moved.*

1.6.11.3. When a turn has ended, all unit states are reset

1.6.11.4. When the game ends a Game over screen will appear and options to return to the menu or quit game will be displayed

*After the game over state is activated, a game over screen appears and all other inputs other than button presses are disabled. The Game Over screen displays the Winner as well as options to return to the menu or quit the game*

1.6.11.5 Animations play throughout the game when units are moving, attacking, or capturing

*Objective was not met due to time restraints*

## End-User Feedback

I interviewed Dylan again after he had played the finished game.

**Does the completed project meet the requirements that you laid out at the beginning?**

*Mostly*

**What requirements haven’t been met?**

*The map isn’t always balanced when creating it and the AI could do with some improvements.*

**Is the game UI clear and intuitive?**

*Yes, I think the UI is generally well designed and easy to understand the function of each button. However, there can be some improvements such as some way of telling when a button has been pressed. I think there should be more prompts as well when an action is not allowed like moving a unit after it has already been moved or if there are no units to attack nearby.*

**How do you find playing against the AI?**

*The AI isn’t very advanced and is quite simple. From my experience playing against it, the AI doesn’t seem to care very much about its units and has no real strategy when moving around. It does seem to play the game well enough that it could prove a challenge for beginner players but if playing against players with higher ability it would be fairly easy to beat.*

**Any other comments to add?**

*I think you should be able to have multiple AI players at once since this would make the game more interesting. It can also be hard to know what the AI has done since the AI’s turn is practically instant so having a delay between each action that the AI performs would be useful.*

## How the program can be improved

### Artificial intelligence

The AI for the game is very simple in nature and can easily be exploited.

Although the AI was implemented into the program, there are many improvements which can be made to it such as increasing the number of variables that are considered in the Winning function:

Enemy Unit Positions and Structures – When considering enemy positions around the map, several factors can affect whether they are in an advantageous position or not. A good example of this is whether there is an enemy factory close to the AI’s HQ. This would put the AI at a major disadvantage as the enemy is able to continuously produce units nearby.

Enemy Unit Types – The type of units that the enemy has is important when determining the position that the AI is in. For example, if the enemy were to have many Mechs and tanks then they would be in a much better position than if the AI were to have many Infantries. The number of units is not enough to determine whether the AI is winning or not.

As well as this, the addition of being able to have more than one AI in the game would improve it a lot and adding delays between AI actions would allow the user to see what actions the AI has performed as noted by Dylan in 5.2.

### Saving/Loading

An improvement to the saving and loading system would be to have multiple save slots. The user would be able to save multiple games at once and then load whichever file they choose. This would allow for greater flexibility when playing with many different people as more than one game could be stored.

### Code Changes

The GameUI class is poorly implemented with all the buttons hardcoded into the game. A more modular approach to this would be to create a button factory or have a file store all the game buttons.

As well as this, there is still a lot of repeated code in IF statements which could be replaced with a method returning a bool instead such as IsVehicle() or IsStructure().

### Online Multiplayer

The game is currently only available for local play, but Dylan suggested that the game should be able to be played online. This would mean significant changes to the code to allow for online use.

### UI and Animations

The UI could be improved in various ways. Mainly with the use of game feedback to the user with prompts as to whether actions are allowed or not as discussed with Dylan in 5.2. Animations would add another layer of depth and entertainment to the game. It would also be a good indicator for when a button press is registered. Unfortunately, this was not able to be implemented due to time constraints.