Thomas Stacey 3146

The Royal Latin School  52131

h446 component 3: Empire Game Project

Contents

[Introduction 1](#_Toc495437297)

[Analysis 1](#_Toc495437298)

[Computational Methods 1](#_Toc495437299)

[Stakeholders 1](#_Toc495437300)

[Existing Solutions 1](#_Toc495437301)

[Common Features 2](#_Toc495437302)

[Sid Meier’s Civilization III 2](#_Toc495437303)

[Sid Meier’s Civilization: Beyond Earth 2](#_Toc495437304)

[Sid Meier’s Civilization VI 3](#_Toc495437305)

[My Approach 3](#_Toc495437306)

[Essential Features 3](#_Toc495437307)

[Limitations 5](#_Toc495437308)

[Requirements 5](#_Toc495437309)

[Success Criteria 5](#_Toc495437310)

[Design 5](#_Toc495437311)

[Breaking Down the Problem 5](#_Toc495437312)

[Grid 5](#_Toc495437313)

[Pathfinding 5](#_Toc495437314)

[Map Generation 6](#_Toc495437315)

[Player 6](#_Toc495437316)

[Unit 6](#_Toc495437317)

[Combat 6](#_Toc495437318)

[AI 6](#_Toc495437319)

[Fog of War 6](#_Toc495437320)

[City 6](#_Toc495437321)

[Opinion 6](#_Toc495437322)

# Introduction

For my H446 Project, I will be making a 4K strategy game similar to the “Sid Meier’s Civilization” Series. The game will revolve around managing an empire of cities against other players and AI using different types of units. I will be building the game in the Unity Engine, which will allow me to easily compile to different operating systems.

# Analysis

## Computational Methods

This problem is able to be solved through computational methods, as it is primarily a video game. Part of the game involves moving units around, which could be done more efficiently by the user by using a “shortest path” algorithm such as the A\* algorithm, allowing less experienced users to avoid terrain penalties they may not notice along the way. Using computers also allows for automation of tasks such as workers and explorers which would need to be done manually if this were a table top game instead.

AI opponents are another feature that requires computers in order to be implemented, without which the game would not be enjoyable, as there would be no objectives and no win conditions.

The problem is also easily decomposed into separate modules, such as map generation, pathfinding, combat and AI. This means that the game can be coded in separate sections, and each module can be iteratively improved without breaking other modules. This means that an iterative, computational approach is valid for this project.

As there is no need for users to know the inner workings of the game, abstraction is required in order for the user to more easily understand it. This means that the user will only see the finished output, not how the game reached it, making it more enjoyable to them.

The game has a defined set of inputs and outputs, meaning that these can be planned for, meaning users cannot break the program. These inputs revolve around pressing specific keys or clicking buttons on the screen, and any undefined inputs will cause no action. This means that a computer can easily turn inputs into outputs, making it amenable to a computational approach.

## Stakeholders

The stakeholders for my project are people who enjoy playing strategy games similar to Civilization. The game will give players a new experience that feels familiar but also has new features allowing for the players to enjoy the project without feeling it is simply a copy of an existing game. I will use a focus group of young adult players, as they make up the majority of gamers, and use a survey to gain people’s opinions on each prototype of the game. This will allow them to shape the project to what they enjoy, thereby making it better for the end user. Through their feedback, I can then alter the project to better suit their needs, and thereby find the balance between challenge and enjoyment so that the game remains interesting but still fun to play.

## Existing Solutions

Existing solutions to this problem include the titles from the “Sid Meier’s Civilization” series, as well as other games such as “Pandora: First Contact”. These solutions employ various differences, but also have many things in common.

### Common Features

There are multiple features which are common to all the existing games that I am looking at. These approaches include turns, so that no player has much of an advantage over another and tiles to separate the map into playable sections. The maps are generally composed of islands similar to our own planet, with different land types to make the game more interesting. There are also different types of units which can move around the map and cities which act as producers for units and targets for capture. Multiple win conditions allow the games to have multiple different playstyles which all have an equal chance of victory. Most games also have a tech tree, which allows the unlocking of new units and other features, allowing the game to progress rather than remaining static.

### Sid Meier’s Civilization III

Civilization III is an older title, and as such contains features that can be improved upon. These include the use of square tiles meaning that all movements are not equal, and land units being unable to cross water on their own. This, while more realistic becomes potentially frustrating for the player, and I believe later games have better solutions. Civ III also allows unlimited units on one tile, which can cause balance issues during wars. Its semi-realistic style has charm, and also allows the game to run on much lower end machines

  
Sid Meier’s Civilization V

Civilization V has a much more realistic style, which, while looking better also means that the computer running it needs to be much more powerful than Civ III. It uses hexagon tiles, a staple of later games of this type as it allows for equal movement in six directions. It also introduces ideologies, a late game feature which allows more specialisation into certain paths.

### http://core0.staticworld.net/images/article/2014/10/2014-10-16_00032-100526286-orig.jpgSid Meier’s Civilization: Beyond Earth

Beyond Earth has a space setting; however has fairly similar mechanics to previous Civilization games. It does add a few unique features however, such as miasma, a tile modifier which causes damage to units, a tech web rather than a tree, which allows for a more customised path, and affinities which each have their own benefits and weaknesses.

### http://cdn2.gamepur.com/images/civilization_6/civilization_6_map_screenshot3.JPGSid Meier’s Civilization VI

Civ VI has a cartoon style, but is more detailed, so the benefits of using this style in Civilization III are lost, making it simply a thematic decision. It adds features such as ‘breakthroughs’, which give a boost to certain researches, and multi-tile cities through ‘districts’, meaning that more thought has to be put into placement, increasing the difficulty for both old players (who may enjoy the challenge), and new players (potentially putting them off)

### My Approach

In my game, I will be taking features from each of the games I have analysed to create a well-rounded final product. I will be using a style like that of Civilization III in order to reduce performance costs, but hexagon tiles in order to create a more balanced game. Land units will be able to cross into water, but will have lower health and attack damage whilst embarked in order to balance. I will integrate breakthroughs into the game to allow for an interesting game, but will not be including districts because they are needlessly complex and therefore off-putting to new players.

I am using an object oriented approach as the game consists of many instances of the same base entity, for example units and cities, which can be specialised but have the same base properties. This means that I can use a parent class to define units, implement it across a range of different types, and instantiate it with different parameters each time. Due to this, an object oriented approach is necessary for the requirements of the project.

C# is a valid language for this project due to the requirements for an object oriented approach and the time constraints as I already have knowledge of the language. For this reason, I am also pairing it with Unity as this provides the base libraries for rendering and the underlying structure, meaning that I do not need to implement features that have been done many times before, allowing me to focus more on the unique features of my project.

The iterative approach allows me to create systems which work and then incrementally improve on them at a later date. This means that I can test improvements in context of other systems, allowing for the game to work better together, and it also allows me to test whether improvements to modules are required, meaning I can focus on adding new systems and updating systems that need it.

## Essential Features

Based on the time requirements for this project, I have selected the most essential core features of the game that will result in playable end product.

|  |  |
| --- | --- |
| Feature | Justification |
| Tile Based | Makes the game more challenging and allows for players to understand movement |
| Hexagon Tiles | Tessellates with equal distance between all tiles |
| Different types of tile (including water tiles) | Makes game more interesting and challenging |
| Map Generation | So that each game is different and maps need not be specified |
| Different Civilizations | Allows for different playstyles and different AI personalities |
| Revolutions | A unique feature that other games do not have |
| Ability to create new Civilizations out of old ones | Enables for any Civilization to have a revolt, not just specifically defined ones. |
| Ability for Civs to merge | Allows for peaceful conquering (may have to comply with other leader) |
| Combat | Allows for wars |
| Territory | Gives goals to the player |
| Units | Allows for the player to interact with the world |

* Tile Based
  + Using Hex tiles as this allows for equal movement cost regardless of direction
  + Different types of tiles
  + Tiles have movement cost
  + Water tiles & ships
* Map Generation
  + Generates a reasonable map
* Different civilizations
  + Each have different abilities so not all the same
  + Player can choose one or choose random
  + AI Players to fill map
* Revolutions
  + Unique Feature
  + Unhappy areas far from capital can revolt
  + Can join another civ or become independent
    - Independent civ needs name
      * Based on area / city name / defined options based on civ succeeds from
    - Independent civ auto declare war unless peaceful independence agreed
* Combat
  + Units can fight each other
  + Element of randomness
* Territory
  + Cities hold territory around them
  + Borders can prevent passage of troops
* Units
  + Different units with different abilities
  + Settler can found cities
  + Worker does work, builds improvements etc.
  + Various military units with different stats
  + “Great People” provide boosts in different stats

I am not going to implement the following because they are non-essential because they can be implemented at a later stage:

## Limitations

Due requirements of Unity, there will be a minimum specification that computers will need to have in order to run the game at an acceptable framerate. I will lower these requirements as much as possible through efficiency savings, but the time available and my knowledge of the inner workings of C# and Unity may mean that the game will be unable to run on extremely low specification machines. The time constrains may also mean I am unable to implement all of my non-essential features, but these are not required, and so should not damage the final product.

A limited audience for testing may also mean that the final product may not appeal to mainstream audiences, but by selecting testers from a wide variety of people, I hope to reduce this limitation as much as possible.

## Requirements

The hardware requirements of this project include a computer of sufficient computing power to obtain an acceptable framerate running Windows, Mac OSX or a Linux distribution. The system will need adequate secondary storage space to hold the game data and some kind of input device so that the user can interact with the game.

## Success Criteria

* TODO

The project will be successful when…

Table

Success criteria / how it will be measured

# Design

## Breaking Down the Problem

The game can be separated into different modules which each can be made separately and replaced with more efficient code later. This means that other modules can be implemented later, and each can be tested separately, removing the need for redundant tests if the code has not been changed.

### Grid

The grid module will be responsible for control over the layout of the game, and will need to contain rendering features so that the user can view the world, a way to update cells when changes to the world happen, and some method of uniquely referring to each cell. It will also contain definitions of types and data about the cells that will allow it to be correctly manipulated by Unity.

### Pathfinding

The pathfinding module will be responsible for traversing the grid in the most efficient way possible. It will need to implement the A\* algorithm, and be able to send paths to units that request it.

### Map Generation

The map generation module will need to take types and parameters about them and use them to create a pseudo-random map that obeys certain rules but will provide a new playing experience each time.

#### Map Generation Algorithm

PROCEDURE GenerateMap(width, height, islandSizeMin, islandSizeMax, numIslands)

FOR i = 0 TO numIslands DO

centreCoords = Random Coordinates WHERE !islandTiles.Contains(this)

numIslandTiles = Random BETWEEN islandSizeMin, islandSizeMax + 1

centreCoords.Type = plains

islandTiles.Add(centreCoords)

possibleTiles.AddRange(centreCoords.Neighbours)

FOR j = 0 TO numIslandTiles WHERE possibleTiles.Count > 0 DO

coords = RANDOM FROM possibleTiles WHERE !allIslandTiles.Contains(this)

islandTiles.Add(coords)

IF coords.Z < tundraHeight - 1 OR coords.Z > height – tundraHeight THEN

coords.Type = tundra

ELSE IF (coords.Z == tundraHeight - 1 OR coords.Z == height - tundraHeight) 50% CHANCE THEN

coords.Type = tundra

ELSE

coords.Type = plains

ENDIF

neighbours = cords.neigbours

neighbours.Remove(islandTiles)

possibleTiles.Add(neighbours)

ENDFOR

allIslandTiles.Add(islandTiles)

FOREACH possTile in possibleTiles

IF possTile.Type == null THEN

possTile.Type = coast

ENDIF

ENDFOR

int numAllIslandTiles = allIslandTiles.Count

FOR i = 0 TO numAllIslandTiles \* fractionHills DO

coords = RANDOM FROM allIslandTiles WHERE !hillTiles.Contains(this)

hillTiles.Add(coords)

ENDFOR

desertTiles = GenerateZones(width, height, desert, fractionDesert, desertSizeMin, desertSizeMax, numAllIslandTiles, allIslandTiles, map)

allIslandTiles.Add(desertTiles)

forestTiles = GenerateZones(width, height, forest, fractionForest, forestSizeMin, forestSizeMax, numAllIslandTiles, allIslandTiles, map)

allIslandTiles.Add(forestTiles)

civStartPoints = getStartPoints(allIslandTiles, numCivs, width, height)

RETURN map

ENDPROCEDURE

### Player

The player module will be responsible for handling inputs of the user and using them to interact with the game in a useful and predictable way.

### Unit

The unit module will be responsible for handling unit interactions with the world. It will also define the various types of unit including settlers, which can build cities, workers which improve tiles and military units for wars.

### Combat

The combat module will handle unit military interactions to calculate the damage taken by a unit and which unit wins the battle (if any).

### AI

The AI module will control the computer players in the game so that the player can play by themselves. It will need to be able to make decisions based on the current situation of the player in an efficient manner so that the player does not have to wait too long until their next turn.

#### Choosing Actions

In order for the AI player to decide what actions to do, it must reference data about itself and the other players in the game. Each AI will have a set of base values that determine the “rest” states of the various parameters, and the AI will need to choose its action based on the different values. For example, more militaristic Civs would have a higher target for the number of military units, certain Civs may prefer building wonders, some may prefer many cities whilst others few. These priorities will range from 0 to 1, with 1 being most important and 0 being the least according to the current state of the game.

#### City Placement

When the AI discovers a place which it thinks is good for a city, it will trigger building of a settler in the city which can get the settler to the position as fast as possible. This will be done by pathfinding from each city to the target to gain the number of turns, and adding this to the number of turns required to complete the production queue. If the AI does not have a city placed by a certain turn, it will look for the best available space to build a city and follow this algorithm.

### Fog of War

The fog of war module will handle which tiles are visible to the player and colour the tiles appropriately. The tiles should be white when undiscovered and a darker version of the colour when out of view distance. Other units should also be invisible outside of the view distance of the player.

### Camera

The camera module will control the various cameras of the game, including the main camera and the minimap camera.

#### Main Camera

The main camera will need to be able to make the terrain ‘loop’ across the horizontal axis, but not the vertical. It will need to limit the user’s view range to how far they have explored.

##### Vertical FOV Calculations

In order to limit the users FOV, there needs to be calculations to calculate the position of the user’s FOV on the world plane. This could be done via raycasts, but would be more efficient and reliable using maths.

As we are given the vertical FOV by default, working out the z position when the camera is facing straight down is fairly simple, as the angle below and above the camera’s z coordinate is half the camera’s FOV. Given the camera’s y coordinate too, , so the absolute coordinates for the top and bottom are . With the default FOV of 60°, rotating the camera 30° from the vertical means that the bottom z coordinate is simply the z coordinate of the camera, and the top is .

As shown in the diagram, the positions of the top and bottom of the screen in the world can be calculated with and where is the rotation of the camera with 0 being horizontal, is the vertical FOV of the camera and all angles are in degrees.

In order to check whether the camera can move in a direction, the program must keep a log of the highest and lowest x and z world positions of the currently discovered tiles and allow movement only while the FOV stays within the area between these.

##### Horizontal FOV Calculations

The horizontal FOV calculations are harder, as if the camera is not facing straight down; the distance in world space between the edges of the screen varies depending on the position on the screen. However, as discovered tiles should always be visible, and undiscovered tiles do not matter, values at the bottom of the screen should be used.

Horizontal FOV is calculated by . However, is the camera’s aspect ratio, and so .

In order to calculate the x distance, we first need the distance from the bottom of the screen to the grid, which is given by . The angle is the same as previously calculated for the vertical FOV. We can then use this to calculate the difference in x to the bottom left and right corners of the screen, which is . This can then be added or subtracted from the x position of the camera to give the positions of the right and left corners respectively.

#### Minimap Camera

The minimap camera will show an overview of the entire area that the user has explored. It will need to zoom out as the user explores more territory to show them where in the world they are.

### City

The city module will control production inside cities, as well as parameters such as buildings and hitpoints. This will allow cities to be captured by enemies and pass data to other modules for use.

### Opinion

The opinion module will define the happiness of the city, which will then be used to control other parameters such as likelihood of a revolution. Happy cities will provide bonuses to the player, whilst unhappy cities will cause debuffs.