CTEC3451 Development Project

Splash X6 Literature review

Jordan Tyler Gray (P2540338)[1]

[1]De Montfort University

June 2, 2022

Abstract

This paper will investigate material surrounding the market in which I am aiming to develop this project in order to create its' foundation. researched is the types of games which are popular with consumers in this area, collecting common features and how they are designed & implemented, alongside and user interface practices.

The information discovered is used to project requirements and user cases, preceding the creation of a plan to validate the implementation of the requirements. The prototype is designed and developed on the foundation of these use cases.

Index

Keywords

Game Development, LibGDX, 4X, Kotlin, Java

Acknowledgements

The development of this report and it's artifacts was made possible by those who supporting the project, and myself. Firstly, in these difficult times of late, I must thank my close friend Courtney Harding alongside my closenit support group for providing me with the mental stamina to push forward with my studies. Without their input, the toll of the times in which we live are likely to have gotten a firm grasp onto the best of me. In no particular order, I also formally thank those for their input on the project itself, no matter how trivial.

Atallah Hezbor for contribution towards the repository's configuration, and initialisation of automated cloud building using GitHub Actions, which led to the possibility of autonomous internal unit testing.

Lewis Gray, my brother, who has provided artistic support & assets and, alongside Thomas Bird & Jamie Watts, has been a resource for spontaneous user testing since the original iterations of this project were proposed over three years gone by.

My university peer, and a good friend, Dylan Brand, who contributed unfathomably to the beginning of the development of the artifact. Providing conceptualisation and design input for the underlying architecture used to build the title, the audio system, and being my go-to for support when conversing through new ideas, troubleshooting, and intramural user testing; All whilst showing overall tremendous interest in the project.

Followed closely by all those who came before him in my journey of study in this field, and by no means to be considered last, my contractual product supervisor Dr. Stuart O'Connor, who takes special interest in areas of artificial intelligence which extend into the gaming domain. He has shown interest in, and support for, the concept of the project since its' proposal and has guided me along my expedition in it's realisation. His recognisably valuable opinion alone provided the validation and comfort that the project showed promise.

My father, Michael Gray, who supported me financially when my computer didn't survive the trip home for Christmas. Without his help I would not have been able to work for the month leading upto the deadlines.

Finally, Arjab Khuman for showing interest and support throughout, even with absolutely no obligation or reward for doing so.

1 Introduction

Thus far, Splash X6 *(abbr. X6)* is a loosely proposed title with vague preliminary goals. Following in the steps of the generations before, the title's genre, features, design, implementation, and more all remain open. The intent in this paper is to define these goals, shaping the future of this game.

2 Market & Literature review

To start, existing titles and literature must be examined. More must be known concerning the desires of users, and the methods of implementation. With extraneous focus on the existing market existing games, and literature to review common or unique features which may be implemented in or influence X6.

2.1 Sid Meier's Civilisation

Civilization is a 4x turn-based strategy game first released in 1991 in which the player supervises a human civilisation throughout history, spanning several hundreds years, in a historically accurate timeline with characters based on world leaders, and civilisations and monuments also based on real counterparts. 2K (2015), Benj (n.d.). The 4x sub-genre game play involves Exploration, Expansion, Exploitation, and Extermination (of all other opposition). This style of game is similar to many of the titles which are to follow.

2.1.1 Units

The user has the responsibility of generating and controlling units, which may be defined as a player control-able entity, or playable character. There exists many types of units, which may realise as a single or multiple humans, or a vehicle. Each of these types has unique attributes, strengths and weaknesses. These differences plays into the game's strategic nature, where the player must both produce appropriate units for the intended usage, and consider how those units are then to be used in order to utilise them to their full potential. (2016*a*)

During game play, units are either idle or have an action which they will perform when the user (or A.I player) ends their turn.

2.1.2 Cities & Production

Upon starting a normal game the player is provided a settler unit which can settle a village, alongside some confrontational units which can be used to defend the settler. These settlements are home to the civilisation's general population (*gen-pop*), whose productivity is determined by the properties of their home. These hubs of productivity can then be put to use by the user in order to have the gen-pop construct structures to expand and enhance the settlement in order to increase it's productivity, or invest that productivity in training units, or into research projects which enable access to further resources. As an interaction, this functions through the user adding projects to a queue, and the city contributing it's production power towards it upon ending a turn (2016*b*), Harbison (2018).

This relationship between cities and units is a crucial part of a 4X game's mechanics, and is a feature in the other games talked about. Taking a more objective view of this system, a settlement is said to have a quantity of productivity, determined by its size and qualities

& similarly, a product (Unit, Structure, Research) has an associated productivity cost. These values determine the length of time required to complete the development (in turns).

If a development has a production cost of 100, and is being worked on within a settlement which can contribute 75 production per turn, then it will take two turns to complete that project. The remainder is contributed to the next product, if one is queued.

$$ceil(\frac{Productioncost}{CityProduction}) = turnsToComplete.$$

2.2 The Battle Of Polytopia

TBoP is a different style of 4X title which shares many of the same features as Civilisation, but is not historically accurate and uses a completely different UI style, aiming to be more modern, simplified, and touch friendly (17). It's 'civilisations' and characters are entirely fictitious. Its games are short by comparison, between 30-100 turns is average depending on the game's configuration, compared to several hundred in an average game of Civilisation. Aiding in this is the size of the world, which is small. The game also has the ability to automatically configure the world size purely based on the number of teams, and the desired difficulty.

2.3 OpenTTD

Open Transport Typhoon Deluxe is an open source remake of Transport Tycoon Deluxe ((2005)). Initially released in 2004 orudge (2004), and maintained to this day (2021). Whilst X6 won't be a transport simulation game, the open nature of this project provides a great repository of information.

2.3.1 Windows

The UI in this game is unique. A custom built framework for creating windows in-game gives the game a more operating-system feel overlaid atop the world (15), as opposed to the other games covered which use dock-style UI elements, where UI is in a decorated container and anchored to the edges of the display, or in floating controls where the UI is placed into the viewport as to appear in the world next to the item the user is interacting with.

2.3.2 Viewports

Another unique feature of this game is its use of multiple *viewports*. A viewport is a region of the screen that is used to render polygons as viewed by a camera in the world (*Viewports & Clipping* (n.d.)). OpenTTD makes use of multiple viewports to allow viewing of different parts of the world at the same time by allowing the user to add more as windows Mdhowe & Librarian (2004), and uses them in tool windows to show the entity the window is referring to ((2008)).

2.4 Common

These games have common features, specifically :

2.4.1 Teams

All of these games have 'teams', in the 4x games considered, these are in the form of other civilisations. In OpenTTD, especially in multiplayer, there are transport companies.

These all give the player something to play against, so they are not alone within the game world. The 4X games' teams have different attributes and abilities, and can either aid or harm the user.

2.4.2 AI

Civilisation and TBoP both support multiplayer, but also contain a custom AI synthesizer for opposing teams which can replace an opposing player, filling spots in an underfilled multiplayer game, or making single player games possible.

2.4.3 path-finding

All of the games mentioned contain entities which must move around the world, and do so by calculating a path to do so. OpenTTD uses a custom solution, but there are open methodologies and implementations which can be of use.

2.4.4 internationalisation

As a standard expectation, they also all support language localisation as a minimum. OpenTTD, being a business simulator, uses currency - it also provides options to localise it's currency.

2.4.5 World generation

The worlds in these titles are generated using custom build systems, each desiring a particular style of world and having different clutter with which to populate the world with.

2.4.6 Fog of war

Civilisation and TBoP contain a fog-of-war style system which hides the world from the players view, forcing them to explore to learn about surrounding resources and their opponents movements.

Civilisation takes this a step further by only displaying areas of the map in the last state they were when they were last explored. This forces the player to scout the world in order to learn of their opponents movements throughout the game.

2.5 Engine

To ease development, the product will be developed in languages with which i am most proficient, Kotlin & Java, leading the project to use a jvm engine. The largest engine for Java is the Light Weight Java Gaming Library (*LWJGL*) which is a wrapper binding java directly to industry standard libraries such as OpenGL, GLFW, Vulkan, OpenCL, and OpenAL ((n.d.*a*)). It's low level, complex, unforgiving, and requires the learning of each of the libraries. It does not provide an engine, but rather the tools to make an efficient engine. LibGDX is an engine built on LWJGL, providing a framework, tools, and handles the complex low-level calls to the libraries, providing a higher-level development environment. It also openly supports the use of Kotlin, seemingly the only engine to do so. LibGDX will be the engine that X6 is developed in.

2.6 User Interface

Looking at the user interface games covered so far, it seems they follow a similar design language. The UI is styled to fit with the eras of the developing world, with window backgrounds textured with yellowed paper and elements and windows following intricate shapes. This is all mixed with more modern looking interactive components and icons for better interaction. The style itself is visually stunning, fitting with the themes of the games. (13) Anno is another large production game which uses a similar UI design, and even though their research is private it can be assumed that it must be desirable with players. (14)

These similarities are found in the modern iterations of these games, however my personal goal for Splash has always been to reminisce the older 'retro' style of game, so instead a closer look at the original version and how they handled the interaction is due.

Civilisation 1's UI was constructed with docked grey windows with simplistic UI elements. A text based menu bar at the top of the game, similar to the 'file, edit' type of menu bar used in many applications, housed many of the options and tools for the player to access. There was little to no complex stylisation. (16)

OpenTTD is similar, it's UI is based almost exclusively on non-docked windows simply styled with solid colors. It also uses a menu bar, but instead uses square buttons and icons, as opposed to text. (15)

X6 will aim to take it's design language in this direction; a windowed UI with simplistic design characteristics, and a top-mounted menu bar to access everything. The unique menu of OpenTTD is preferred as it differentiates itself from the kinds of menu's seen in productivity applications which aim to tuck lesser used options out of the way, and instead places game-play-dependant activities front and center.

A abstract window based interaction system is very flexible, being able to encompass almost any feature in a standardised fashion as opposed to having to create a uniquely shaped complex area for controls like in Civilisation and Anno. This flexibility and reduction of complexity would ease development, so it's the preferred option.

Going with a retro style windowed UI, a style can be created for the engine. LibGDX requires a Skin (badlogic (2013)), and there is a tool called *TextComposer* which can be used to design and create them (raeleus (2021))

2.7 World generation

All of the games covered have the ability to procedural generate a world so that the player has unique terrain for each game. Travis Archer writes how specific kinds on noise can generate pseudo-random landmass and height maps (Archer (n.d.)), and Jindrich Mor´vek delves into great depth about generating terrain using noise, erosion, and notably tectonic plates using Voronoi diagrams (Mor´vek & Kovalcik (2011)). These same kind of diagrams may also used to generate world regions representing different biomes.

2.7.1 Noise

Ken Perlin created Perlin noise, an industry standard gradient noise (Perlin (n.d.)). It's useful for generating landmass and height maps, but it's not the only noise algorithm. Perlin continued on to create Simplex noise to counter efficiency and artifact (Perlin (2001)). Perlin filed patents for his noises, creating usability issues, especially for using Simplex, as a result OpenSimplex was created. To avoid liability issues, this will be the one X6 uses by default, a java implementation of which is found in the FastNoise Lite Library (Peck (2021)). This library also supplies other noises, so options for the user to switch which noise is used could theoretically be added.

Word count: 1932

3 Functional Requirements

3.1 Users

The intended audience of this game is those who enjoy strategic 4X games, those who think rationally and turn by turn as opposed to players who seek adrenaline from real-time game play, or armchair gamer solving puzzles.

3.2 feature requirements

Collating the features seen in existing, similar, titles, I have created a list of features that I would like to see implemented into X6. These desires form the basis of requirements for the prototype. Later in the project, the product will be compared against these functional requirements to test if it was successful.

- World generation.

  Having worked with world generation before, I believe it to be perfectly possible to create a world generation system to enhance the worlds in which the users play in a similar fashion to those seen in the games investigated.

- Language Internationalisation

  I'm a firm believer that software should be accessible, not limited to those who are in the same locale as those who made it. As a minimum, the game should support the ability to switch languages.

- Civilisation style teams of units

  Where a player has an array of units at their disposal, and can produce more. A world can contain multiple teams, AI or other players which is composed of these units

- Fog Of War

  Aiming for a 4X style game, the player needs a reason to explore. At minimum, i'd like the game to only show areas of the world the user has explored.

- Pathfinding

  For more advanced unit movement, I'd like to implement some form of pathfinding, such that a destination may be selected, and the unit will figure out how to get there by itself.

- Settlements & Production

  Crucial to expansion, and playing largely into how many of these 4x games work, there must be some kind of production system. X6's will revolve around settlements (cities), in a similar fashion to TBoP & Civilisation.

- Multiplayer At a stretch for the prototype, the ability for clients to pass back and forth the game data at the end of each turn to make lan multiplayer possible.

- Turn Based All actions occur on a turn-by-turn basis, mimicking a table-top board game.

- Save / Load The games produced will be longer scale, and thus likely played in multiple sessions, requiring that the user is provided some way to save to and load from the disk.

- Music For ambience, the game will play some music whilst in-game.

4 Use cases

#	Use Case	Description.	Actions involved
UC001	Develop settlements	A player claims land as thier own to build a village which is grown into a town, and then a city.	Creating a settler, travelling to a destination, claiming land, developing supporting structures, defending it.
UC002	Attack an enemy	Units approach an enemies units or settlement and attack it.	Travelling, attacking, defending.
UC003	Build an army	The player produces various kinds of units to build an army to protect thier land and overthrow thier opposition.	Cities produce units. Units upgrade with experience.
UC004	Explore the map	The player cannot see the entire map, and has units travel around to discover areas in order to make them visible.	Exploring, observing
UC005	Observe other players	When exploring, the player can observe opposition's movements, settlements, and units. Knowing where they are located, gathering or traveling can give them time to prepare to defend.	Exploring, observing
UC006	Determine threat level	From observing the other players, a player can determine if they are under immediate threat from attack, or if they are potentially unprepared to defend.	Exploring, observing
UC007	Defend a settlement	Units can be stationed near a settlement in order to prevent an attack from damaging it by retaliating.	Producing units, stationing them near a settlement.

Table 1: Use cases

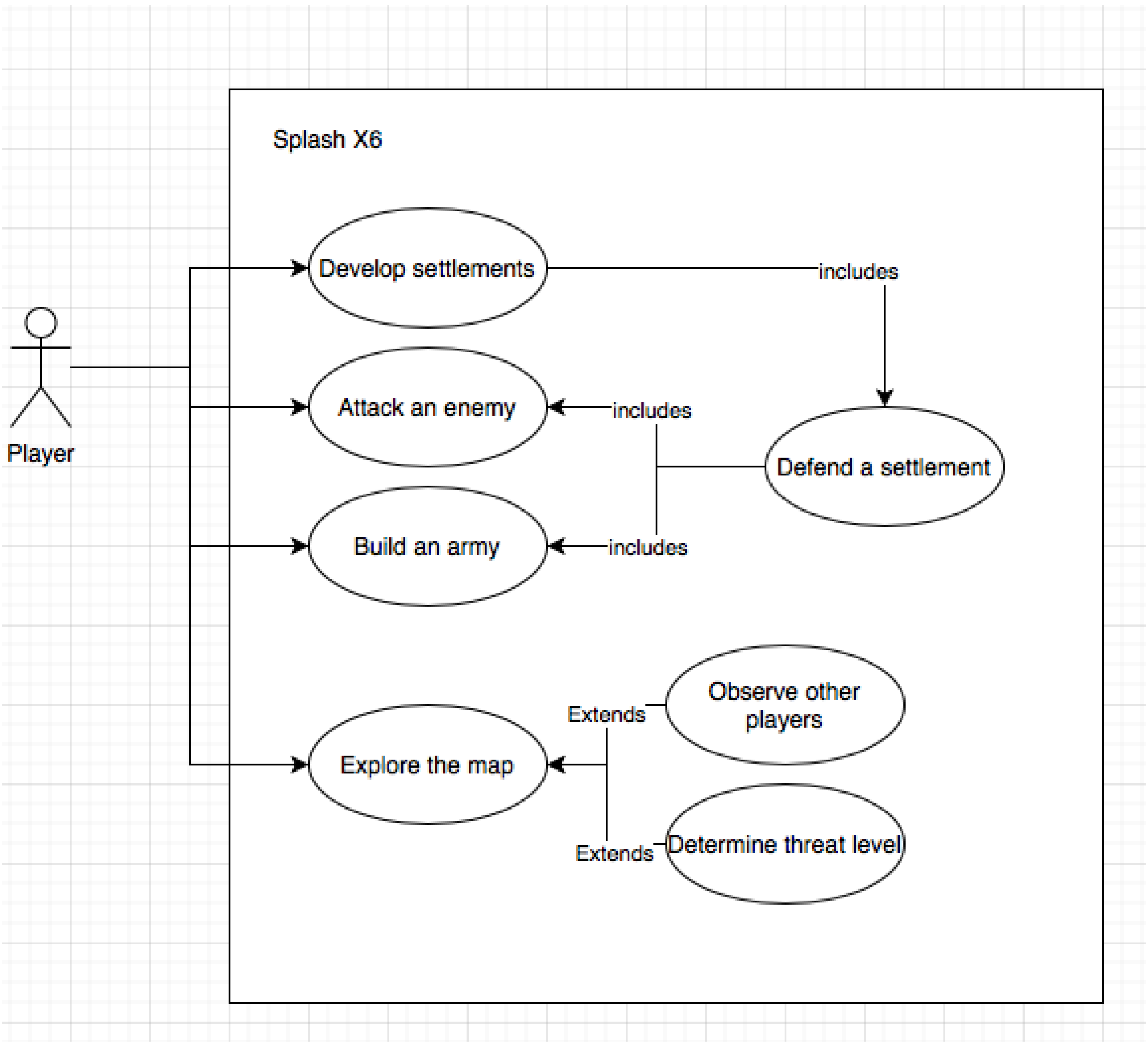


Figure 1: Use Case Diagram

5 Test plan

A game is developed to be user centric, and as such it should be tested from a user's point of view. The testing of this prototype will stem from a mix of unit testing and user testing. The first will assert that individual sections of the system are, to some degree, stable and operate successfully in accordance to how they are expected to function. Unit testing is performed in the eye of the developer to ensure functionality. User testing, on the other hand, involves passing a prototype copy to play test and providing a method for them to return detailed and controlled feedback regarding their experience.

With access to peers and family to act as play-testers, the game can be tested for feedback against a set of criteria by requesting them to try specific features or perform specific actions, or they may report other issues or improvement suggestions that they encounter.

5.1 User testing

With user testing, it's important to understand that they are likely not to have development experience. Feedback must be controlled to ensure that useful and detailed information is obtained that's actually useful in improving the product.

Obtaining the prototype is achieved through the git-hub repository, which it will be assumed they have no experience so a step-by-step walkthrough of how to obtain the latest cloud build was created (Gray (2021)), and play testers will be linked to. This can be updated with further details on how to provide feedback. Along with this they will be provided a link to a feedback form, and list of features or actions to try, as mentioned before, to have them test specific areas.

The feedback form will coax the tester into providing specific details to help efficiently extract the maximum of useful details. The created form is appended as a PDF, but it can be better explored here : Gray (2022)

5.2 Unit testing

Unit testing is scripted, automated testing performed on the system. For X6, a collection of unit tests will be built in which will automatically be executed when a cloud build is performed (on every commit to the master or main development branch). This will provide historical records of testing, and the artifacts of these cloud builds will be the main method of non-developers to obtain the prototype.

For a selection of the major features a range of tests will be created intended to simulate the use of that code, and the resulting values or states will be checked against what is expected to determine if the system functioned as was expected.

Unit tests are appended : Table 2

6 Design Documents

n.b design graphics are appended, instead of in-line.

6.1 World generation

6.1.1 Voronoi Biome Generation

An open implementation of a Voronoi diagram will be used to implement biome generation, where each Voronoi cell is mapped to different kind of biome, which then defines what kind of terrain, foliage, etc will be generated in that area.

6.1.2 Stages

The world generation will take place in stages, where each stage is world generation behaviour which is isolated. They iterate through the terrain with a different effect. There must be a stage prior to all others to generate the 'base' terrain (land mass & water), then other stages which modify it in some way. Figure 3 shows the type hierarchy used to define these stages, and defines a handful of different stages. The final stage generates navigation data layer required for pathfinding. To implement, the output of one stage is fed directly to the input of another, working through the list of stages (4). The generation sequence *could* be hard-coded this way, feeding one stage into another (5), but an abstract dynamic system can be used to process any list of stages, in any order (6).

6.1.3 Tile Interpolation

To make the world more visually appealing, each individual tile will be 'interpolated', blended, with those around it - reducing the contrast between tiles of different kinds. A generation stage can stop on each isometric tile and check the two tiles adjacent to it's vertices. If the two tiles are the same, the corner of that vertex takes on the texture of the surrounding tiles (9, 10).

The assets being used (FreeCiv's ampillo, (n.d.*b*)) contains an entire sprite sheet for each possible tile, filled with images of itself merged with all other tiles. It references the desired tile by name, using the format *top_right_bottom_left*, where each value is a letter representing the kind of tile (d = deep ocean, s = shallow, l = land, etc.) displayed on the corresponding corner. The generation stage mentioned above simply must generate a resource name in this format.

(See 8 (Without), & 7 (With))

6.2 Save / Load

If all game data is stored in a single serializable object, then that object may be saved to disk using a process called serialization, where the entire object in it's current state is dumped. Loading this file again is called deserialization, and results in an object instance, just as it was when it was saved, than can be used in the client. (11)

This is a simple approach, and will be what X6 uses.

6.3 Multiplayer

An extension of Save / Load (6.2), multiplayer will allow multiple clients to join the same game. It will send the entire GameData object between the clients using serialization; this way the same infrastructure can be re-used and reduces the need to develop the game to be fully server based, reducing complexity. This method is slower, but this can be justified by the game being turn-based, not real time. Figure 11 (12) shows the intended data-flow of the application, and Code 1 (7) displays how this flow would function.

6.4 Production

The production system will revolve around a settlement and will function as discussed in 2.1.2, using the UI described in (2.6); The user will open a window from the dock where they can select a settlement to manage, where they can then alter which production projects are in it's queue.

They'll also need to be able to view the production power of the city, and the cost of production projects.

As described in 2.1.2, the relationship between the production power and the cost will determine how many turns it will take to complete a project, and obtain its product (18, 19).

7 Implementation report

- Created a LibGDX project
- Used a LibGDX tiledMapRenderer to a tmx tilemap created in Tiled.
- Created 'GameHypervisor' and 'GameData' objects, static containers with calls to perform actions within game
- Added ability to load the tilesets used in the tmx, and created a plist containing the name of every visual asset's name which will be used to load and reference them internally.
- Created a player sprite which spawns in the world.
- Extended LibGDX's window, implementing an in-game-window with custom features.
- Created a simple menu and splash screen
- Created a basic world generation test, to see if it was possible to internally populate a LibGDX TiledMap object, was only designed to load a pre-designed tmx file. It worked!
- Started a simple world generation system with noise, random foliage & player spawn.
- Made a menu to access windows and options
- Created windows to manage units.
- Implemented unit's ability to perform an action by defining scripts and an action dictionary. Actions are filtered by the unit's class. The settler can now settle a city.
- Created an extension for the camera, it now moves smoothly using a 'desired position' system and delta-time. It calculates positioning and angles in order to focus on a target.
- Created a better key input system, inputs are now handled dynamically using bindings.
- Added in-game music, and ability to connect to and manipulate the Spotify API using a java wrapper.
- Added dynamic pathfinding using A*
- Created an options window which is dynamic and uses reflection to alter properties.
- Added 'fog-of-war', only some tiles are rendered.
- Added facility to perform CI/CD cloud unit testing.
- Started work on networking
- Created a state-machine for future use in opponents.
- Started a production system for settlements.

Word Count : 292

References

(2005).
URL: *https://www.openttd.org/about*

(2008).
URL: *https://wiki.openttd.org/en/Manual/Tutorial/Busesgiving-orders-to-your-bus*

(2016*a*).
URL: *https://civilization.fandom.com/wiki/Category:Units_(Civ6)*

(2016*b*).
URL: *https://civilization.fandom.com/wiki/Production_(Civ6)*

(2021).
URL: *https://www.openttd.org/downloads/openttd-releases/latest*

(n.d.*a*).
URL: *https://www.lwjgl.org/learn-more*

(n.d.*b*).
URL: *https://freeciv.fandom.com/wiki/Amplio*

2K (2015), 'Civilisation i'.
URL: *https://web.archive.org/web/20150918012513/https://www.civilization.com/en/games/civilizatio i/*

Archer, T. (n.d.), 'Procedurally generating terrain'.
URL: *http://micsymposium.org/mics_2011_proceedings/mics2011_submission_30.pdf*

badlogic (2013), 'Skin · libgdx/libgdx wiki'.
URL: *https://github.com/libgdx/libgdx*

Benj, E. (n.d.), 'The history of civilization'.
URL: *https://www.gamasutra.com/view/feature/1523/the_history_of_civilization.php?print=1*

Gray, J. (2021), 'Getting the latest build · shinkson47/unix6 wiki'.
URL: *https://github.com/Shinkson47/UniX6*

Gray, J. (2022), 'Splash x6 prototype negative feedback'.
URL: *https://forms.gle/AmZdyK7RduZJMvKD8*

Harbison, C. (2018), 'Civ 6 early game guide: Beginner strategy and tips for gaining your first victory'.
URL: *https://www.player.one/civ-6-guide-early-game-strategy-civilization-vi-beginners-tips-districts-123087*

Mdhowe & Librarian (2004), 'Openttd | extra viewport'.
URL: *https://wiki.openttd.org/en/Manual/Extra%20Viewport*

Mor´vek, J. & Kovalcik, V. (2011), Planet generation: Mimicking the nature's way, *in* '2011 International Conference on Computational Science and Its Applications', IEEE, p. 24–32.
URL: *https://ieeexplore.ieee.org/abstract/document/5959558?casa_token=5bQmHhjecwkAAAAA:l1r_ cWI7df*

OpenTTD (2012).
URL: *https://wiki.openttd.org/en/Manual/Tutorial/Busesgiving-orders-to-your-bus*

orudge (2004), 'I proudly present to you openttd! - transport tycoon forums'.
URL: *https://www.tt-forums.net/viewtopic.php?t=6559*

Peck, J. (2021), *FastNoise Lite*, FastNoise Lite.
URL: *https://github.com/Auburn/FastNoiseLite*

Perlin, K. (2001), 'Noise hardware (chapter 2)'.
URL: *https://www.csee.umbc.edu/ olano/s2002c36/ch02.pdf*

Perlin, K. (n.d.), 'Us2002135590'.
URL: *https://worldwide.espacenet.com/publicationDetails/biblio?FT=Ddate=20020926DB=locale=en_*

raeleus (2021), *LibGDX Skin Composer*.
URL: *https://github.com/raeleus/skin-composer/releases/tag/46*

Viewports & Clipping (n.d.).
URL: *https://www.tecgraf.puc-rio.br/ftp_pub/lfm/L1J_ WindowViewport.pdf*

.

Appendices

```
// Pseudo-server.

// Creates a LAN port open to accept new clients
syncronized fun hostOpenPort() {
    ss = ServerSocket(PORT)
    while (acceptingClients) {
        client = ss.accept() as Client
        clients.add(client)

        // Notify client that they were accepted into the game.
        client.joined()
    }
}

// Stop accepting new players and create game.
fun closePort() {
    acceptingClients = false;
    createGame()
    pushToAllClients()
    firstTurn()
}

fun pushToAllClients() {
    clients.foreach {
        it.setGameData(server.GameData)
    }
}

fun firstTurn() = randomClient().yourTurn()

// When any client ends thier turn.
fun onTurn() {
    // Update sever with changes from the turn just completed.
    server.GameData = currentClient.getGameData()

    // Push the changes to all the clients.
    pushToAllClients()

    // Work out whose turn is next
    val nextClient = clients.next()

    // Notify, permitting that client to play.
    // Once complete, they'll notify the server. Repeat.
    nextClient.yourTurn()
}
```

Code 1: A pseudo-server showing the data flow for multiplayer.

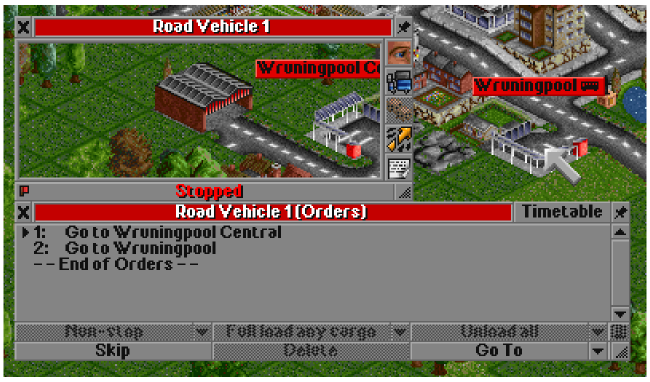


Figure 2: A game window showing a unique camera view within it OpenTTD (2012)

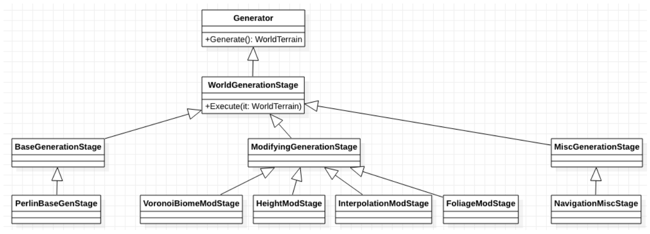
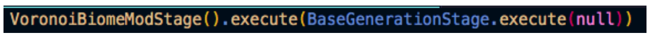
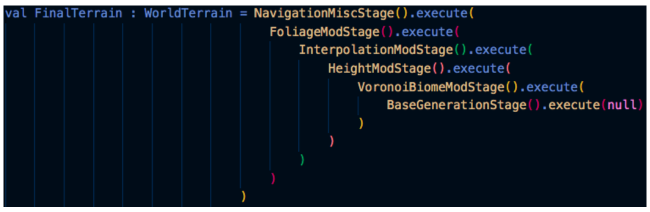


Figure 3: Default world generation stages in order (ltr), and the type hierarchy.



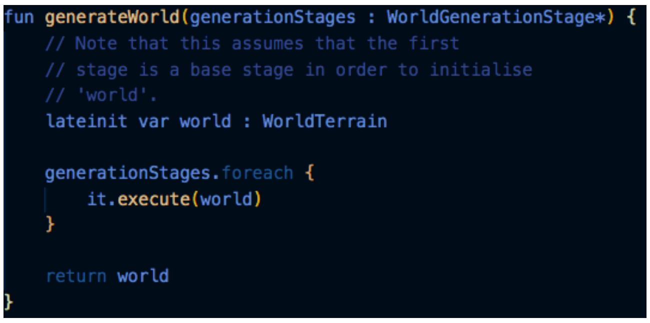
```
VoronoiBiomeModStage().execute(BaseGenerationStage.execute(null))
```

Figure 4: Pseudo-code showing how a stage's output is fed to the next stage.



```
val FinalTerrain : WorldTerrain = NavigationMiscStage().execute(
                                      FoliageModStage().execute(
                                          InterpolationModStage().execute(
                                              HeightModStage().execute(
                                                  VoronoiBiomeModStage().execute(
                                                      BaseGenerationStage().execute(null)
                                                  )
                                              )
                                          )
                                      )
                                  )
```

Figure 5: Pseudo-code showing how 4 could be strung to generate a complete world.



```
fun generateWorld(generationStages : WorldGenerationStage*) {
    // Note that this assumes that the first
    // stage is a base stage in order to initialise
    // 'world'.
    lateinit var world : WorldTerrain

    generationStages.foreach {
        it.execute(world)
    }

    return world
}
```

Figure 6: Pseudo-code showing a dynamic function that uses a list of stages to generate a complete world.

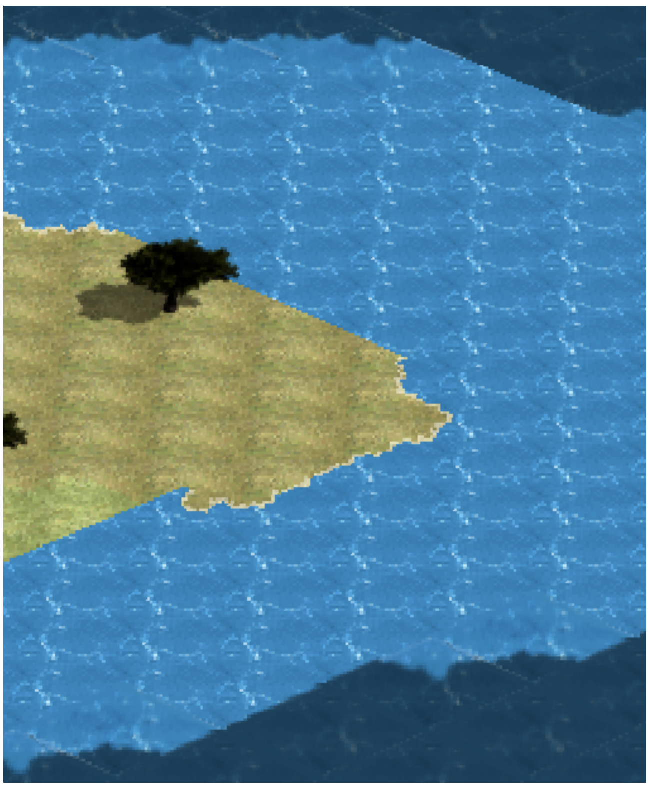


Figure 7: Tiles that have been blended

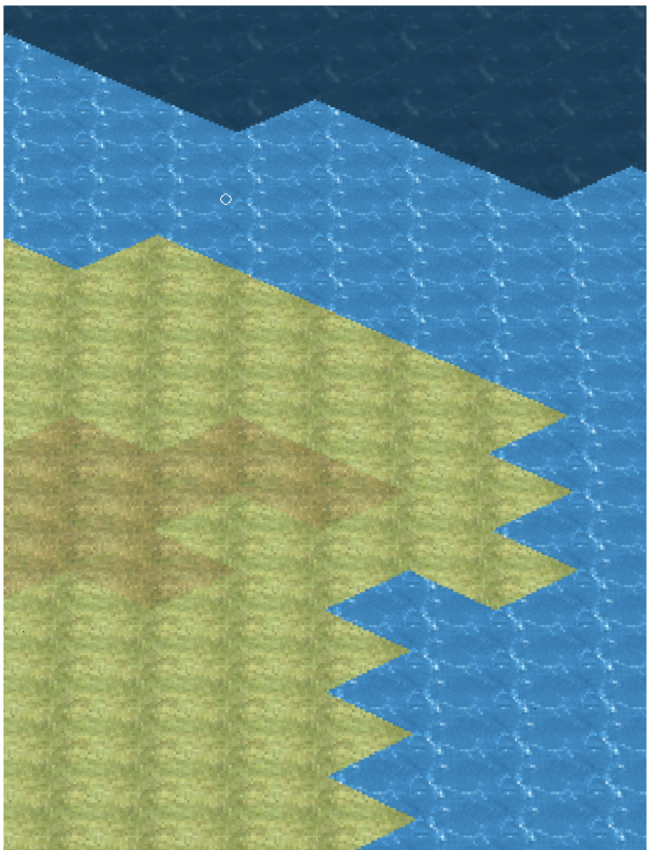


Figure 8: Tiles that have not been blended

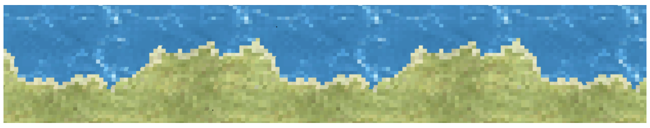


Figure 9: Grass tiles having their corners replaced with water, and water having their corners replaced with grass.

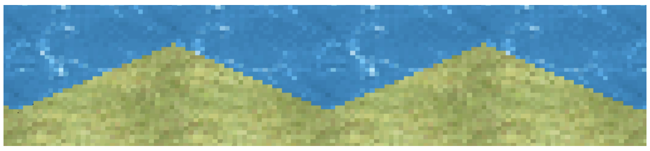


Figure 10: 9 prior to blending

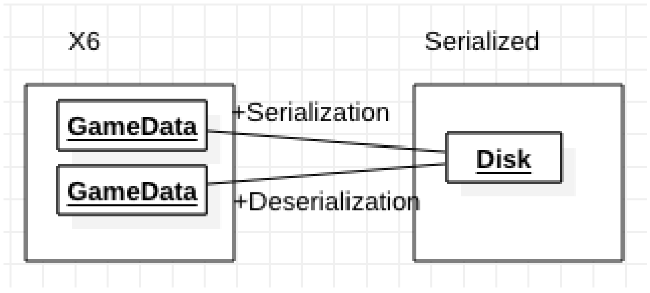


Figure 11: The dataflow for saving and loading using serialization.

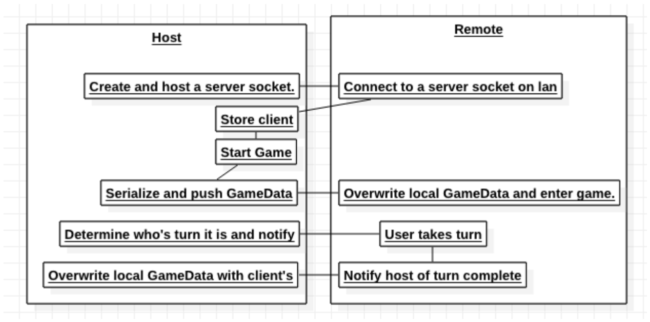

Host
Create and host a server socket.
Store client
Start Game
Serialize and push GameData
Determine who's turn it is and notify
Overwrite local GameData with client's
Remote
Connect to a server socket on lan
Overwrite local GameData and enter game.
User takes turn
Notify host of turn complete

Figure 12: The dataflow for multiplayer using serialization.

Figure 13: A screenshot of Civilisation 6 showing it's UI.

Figure 14: A screenshot of Anno 1800 showing it's UI.

Figure 15: A screenshot of OpenTTD showing it's UI.

Figure 16: A screenshot of Civilisation 1 showing it's UI.

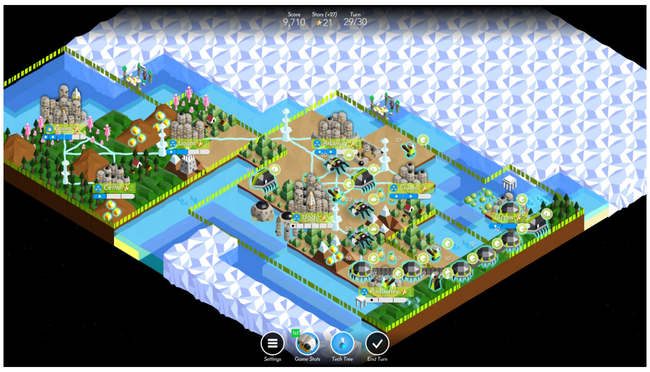


Figure 17: A screenshot of The Batle of Polytopia showing it's UI.

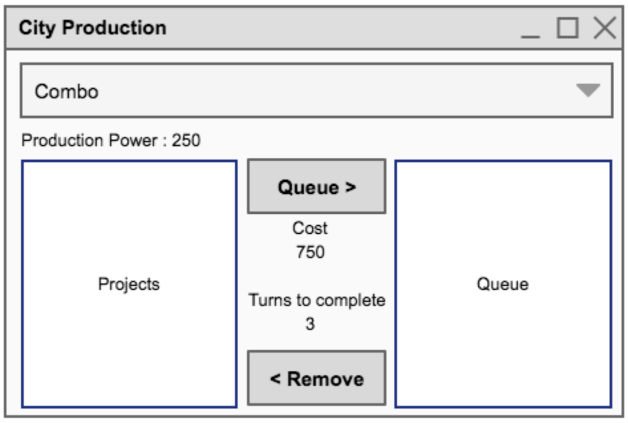


Figure 18: A wireframe for a window providing control over a cities production.

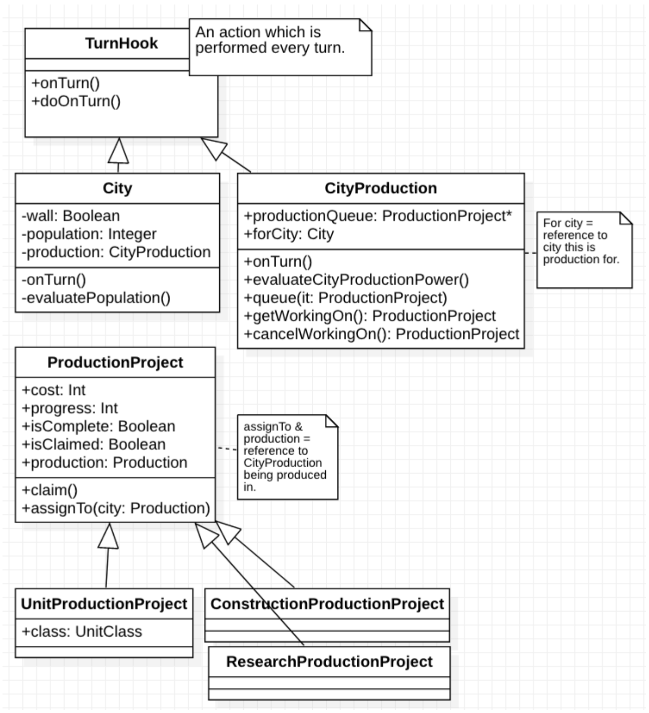
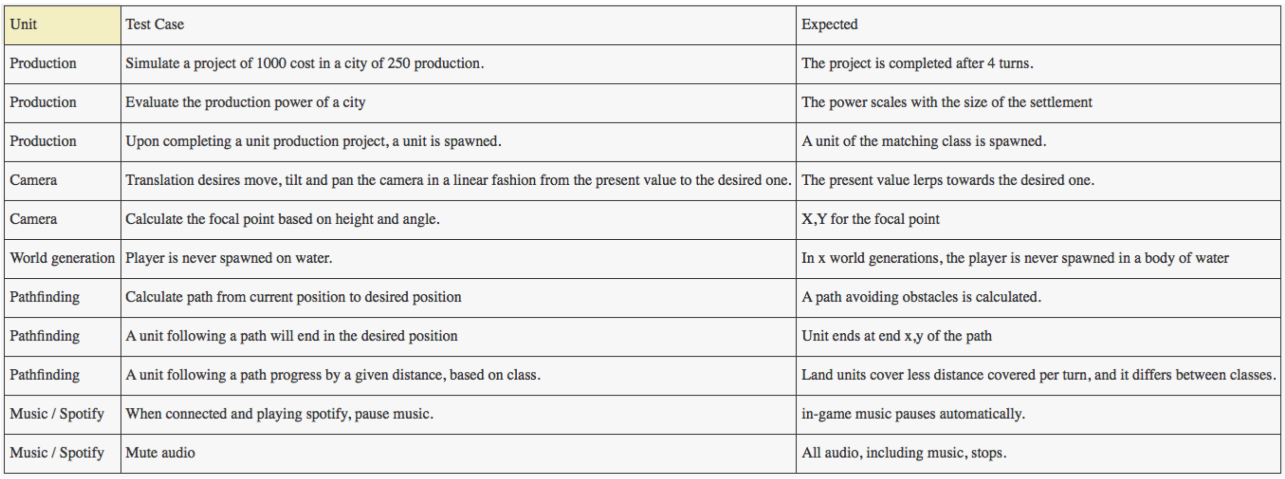

TurnHook
+onTurn()
+doOnTurn()
An action which is performed every turn.
City
-wall: Boolean
-population: Integer
-production: CityProduction
-onTurn()
-evaluatePopulation()
CityProduction
+productionQueue: ProductionProject*
+forCity: City
+onTurn()
+evaluateCityProductionPower()
+queue(it: ProductionProject)
+getWorkingOn(): ProductionProject
+cancelWorkingOn(): ProductionProject
For city = reference to city this is production for.
ProductionProject
+cost: Int
+progress: Int
+isComplete: Boolean
+isClaimed: Boolean
+production: Production
+claim()
+assignTo(city: Production)
assignTo & production = reference to CityProduction being produced in.
UnitProductionProject
+class: UnitClass
ConstructionProductionProject
ResearchProductionProject

Figure 19: UML for a city settlement and it's production manager



Unit	Test Case	Expected
Production	Simulate a project of 1000 cost in a city of 250 production.	The project is completed after 4 turns.
Production	Evaluate the production power of a city	The power scales with the size of the settlement
Production	Upon completing a unit production project, a unit is spawned.	A unit of the matching class is spawned.
Camera	Translation desires move, tilt and pan the camera in a linear fashion from the present value to the desired one.	The present value lerps towards the desired one.
Camera	Calculate the focal point based on height and angle.	X,Y for the focal point
World generation	Player is never spawned on water.	In x world generations, the player is never spawned in a body of water
Pathfinding	Calculate path from current position to desired position	A path avoiding obstacles is calculated.
Pathfinding	A unit following a path will end in the desired position	Unit ends at end x,y of the path
Pathfinding	A unit following a path progress by a given distance, based on class.	Land units cover less distance covered per turn, and it differs between classes.
Music / Spotify	When connected and playing spotify, pause music.	in-game music pauses automatically.
Music / Spotify	Mute audio	All audio, including music, stops.

Table 2: Test Cases