Dragon Skulle Team Project 2020/21

Oscar Lindenbaum - 2005912
Craig Wilbourne - 2045438
Aurimas Blažulionis - 2069871
Harry Stoltz -2035366
Leela Muppala - 2069316
Nathaniel Lowis - 2019185

Contents

1. Introduction

Hex Wars is a real time strategy game set in the medieval period, where kingdoms expand and attack each other in order to conquer the world. Inspired by games such as Civilisation and Risk, our take on an RTS revolves around protecting one's vulnerable capital while trying to conquer the capitals of other players, all via placing and attacking buildings. Furthermore, the fog of war encourages players to explore the map, risking running into other kingdoms and starting a battle for survival.

The game is fully networked, allowing players to go head-to-head against each other and AI over the internet. It is also entirely 3D rendered, allowing players to view the map in fine detail and providing an overall polished look.

2. Requirements

Right at the start of the project, we decided that we needed a detailed set of requirements to help lay solid foundations for our game. This allowed us to gain a strong understanding of the main concepts and features we wanted to implement as a group- helping to ensure consistency as we all could work towards clearly defined goals.

As our vision for the game grew more ambitious, we also added a couple of additional requirements. We always made sure to discuss additions as a group, so that the whole team stayed informed. We also checked that any additional requirements would not interfere with any preexisting requirements. This allowed us to adapt to new possibilities, while also maintaining a clear direction for the project.

Key		
M	Must	Needs to be included to have a functioning product.
S	Should	Adds key functionality (but the product will still function without).
C	Could	Further extends functionality.
W	Want	Would be nice to have but not necessary.

Map		
1	The system shall display a map formed out of hexagonal tiles.	M
2	1. The system shall only display visible regions of the map (fog of war).	S
	2. The system shall update the fog of war when a building is placed.	M
	3. The system shall update the fog of war when building ownership changes, or the building is destroyed.	C
3	1. The system shall utilise a basic map.	M
	2. The system shall generate a random map consisting of different types of hexagon tile.	C
4	1. The system shall place a capital building for each player on the map, at a random location.	M
	2. The system shall ensure each player's capital city building is reachable by opponent players.	S
	3. The system shall ensure the players' capitals are sufficiently evenly distributed around the map.	S
5	The system shall allow the user to move around the map.	S
6	The system shall allow the user to zoom in and out of the map.	C
7	The system shall claim a 1 tile radius by the player around each building.	S

8	The system shall have normal land tiles (that can have buildings placed on them).	M
	The system shall have water tiles (that cannot have buildings placed on them).	S
	The system shall have mountain tiles (that cannot have buildings placed on them.)	C
9	1. The system shall display a variety of additional 3D props on the land tiles surrounding buildings.	C
	2. The system shall display a variety of additional 3D props on the non-land tiles surrounding buildings.	W

Building Placement		
1	The system shall allow the user to enter ‘Place Building’ mode.	M
2	1. The system shall display the valid tiles that can be built on.	S
	2. The system shall display the tiles that cannot have buildings placed on them.	C
	3. The system shall forbid buildings being placed directly on top of other buildings	M
	4. The system shall forbid buildings being placed outside of the visible region of the map (fog of war).	M
	5. The system shall forbid buildings being placed on any claimed tiles.	M
	6. The system shall forbid buildings being placed directly adjacent to each other.	M
3	The system shall allow a basic building with no increased stats to be placed.	M
	The system shall display a shop for placing pre-upgraded buildings.	S
	The systems shall display the cost of placing each building.	S
4	The system shall subtract the cost of a building each time that specific building is placed.	M

Selecting a Building (UI)		
1	The system shall allow the user to select a building they own.	M
2	1. The system shall provide the player with the option to attack an opponent building.	M
	2. The system shall provide the player with the option to increase the stats of the building.	S
	3. The system shall provide the player with the option to sell the building.	C

Attack an Opponent		
1	The system shall allow the user to attack opponent buildings.	M
2	The system shall subtract the cost of the attack from the attacker.	S
3	The system shall have an attack cooldown, so the same player cannot repeatedly spam attacks.	C
4	1. The system shall highlight buildings that can be attacked (buildings inside the range of attack and that	S

	are visible).	
	2. The system shall display the attack cooldown.	S
	3. The system shall display the cost of the attack.	M
5	1. The system shall display a simple attack animation to the user and opponent.	C
	2. The system shall play an attack audio prompt to the user and opponent.	S
6	1. The system shall calculate whether the attack succeeded or failed.	M
	2. The system shall utilise the player's building attack stat and the opponent building's defence stat to calculate how successful the attack is.	S
	3. The system will utilise an algorithm that includes random number generation to add a non-deterministic nature to attacks.	M
7	The system shall award the ownership of the opponent building to the attacker, on a successful attack.	M
8	The system should allow each building to have a certain amount of health, with failed attacks deducting from this health.	W

Increase Building Stats		
1	The system shall allow the player to increase the stats of their buildings.	S
2	1. The system shall allow the player to change their building's stats for a cost.	M
	2. The system shall ensure the higher stats a building has, the higher the cost of stats for that building.	S
3	1. The system shall allow the player to change: token generation rate.	S
	2. The system shall allow the player to change: attack strength.	S
	3. The system shall allow the player to change: defence strength.	S
	4. The system shall allow the player to change: view distance.	W
	5. The system shall allow the player to change: attack range.	W
4	The system shall change the type of building displayed based on the stats of a building.	C

Selling a Building		
1	The system shall allow for buildings to be sold for a partial refund.	C
2	The system shall remove the building being sold.	M

Token Generation		
1	1. The system shall allow for each player to generate a certain amount of tokens every few seconds.	M

	2. The system shall provide the number of tokens based on the token generation rate of that player's buildings.	S

Losing a Game		
1	1. The system shall cause a player to lose if their capital building is lost.	M
	2. The system shall prevent a player who has lost from performing any actions that affect gameplay (placing buildings, attacking opponent buildings or increasing building stats).	M
	3. The system shall allow the user to leave the game once they have lost.	C
2	The system shall remove all of the losing player's buildings.	C

Winning a Game		
1	1. The system shall end the game when all capital buildings have been destroyed, except for the winning player's capital building.	M
	2. The system shall display a victory message to the winning player.	M
	3. The system shall allow the user to leave the game once they have won.	S

AI		
1	The system shall allow for simple AI players that can perform the same actions as human players (place buildings, attack opponent buildings and increase building stats).	M
2	The system shall have AI players that act probabilistically. These will perform different actions based on predefined probabilities.	M
3	The system shall have AI players that use pathfinding to plan routes to attack opponent players.	C
4	The system shall allow AI players to act both probabilistically and utilise pathfinding.	C
5	The system shall ensure AI players prioritise attacking opponent capitals.	S
6	The system shall allow AI players to become more focused on attacking opponent capitals as the game continues.	W
7	The system shall allow only the server to control AI players.	M

Menus (UI)		
1	1. The system shall display a start menu.	M
	2. The system shall display an option to play the game.	M
	3. The system shall display an option to edit sound settings.	S
2	1. The system shall display a pause menu when in game.	S

	2. The system shall prevent the user who opened the options menu from interacting with the game while the menu is open.	M
	3. The system shall display an option to resume playing the game.	M
	4. The system shall display an option to edit settings.	C
	5. The system shall display an option to quit the game.	S
3	1. The system shall provide an interface to upgrade a building's stats.	S
	2. The system shall provide an interface for selecting a building type to place.	S
	3. The system shall provide an interface for attacking a building.	S

Rendering		
1	The system shall allow for perspective rendering.	C
2	1. The system shall allow for game objects to be represented by basic 3D shapes.	M
	2. The system shall allow for game objects to be represented by complex 3D meshes.	C
3	The system shall allow for text to be displayed.	S
4	The system shall use physically based rendering.	C

Networking		
1	1. The system shall allow the user to host a multiplayer game.	M
	2. The system shall allow the user to specify the number of players before a game starts.	W
	3. The system shall allow the user to specify the size of the map before the game starts.	W
2	The system shall allow the user to join a multiplayer game.	M
3	The system shall allow the user to select a game to join from a list of available hosts.	W
4	The system shall fill a game with AI players, if there are not enough human players.	M
5	The system shall disconnect the clients if the host ends the game.	S
6	The server shall manage the authoritative game state.	M

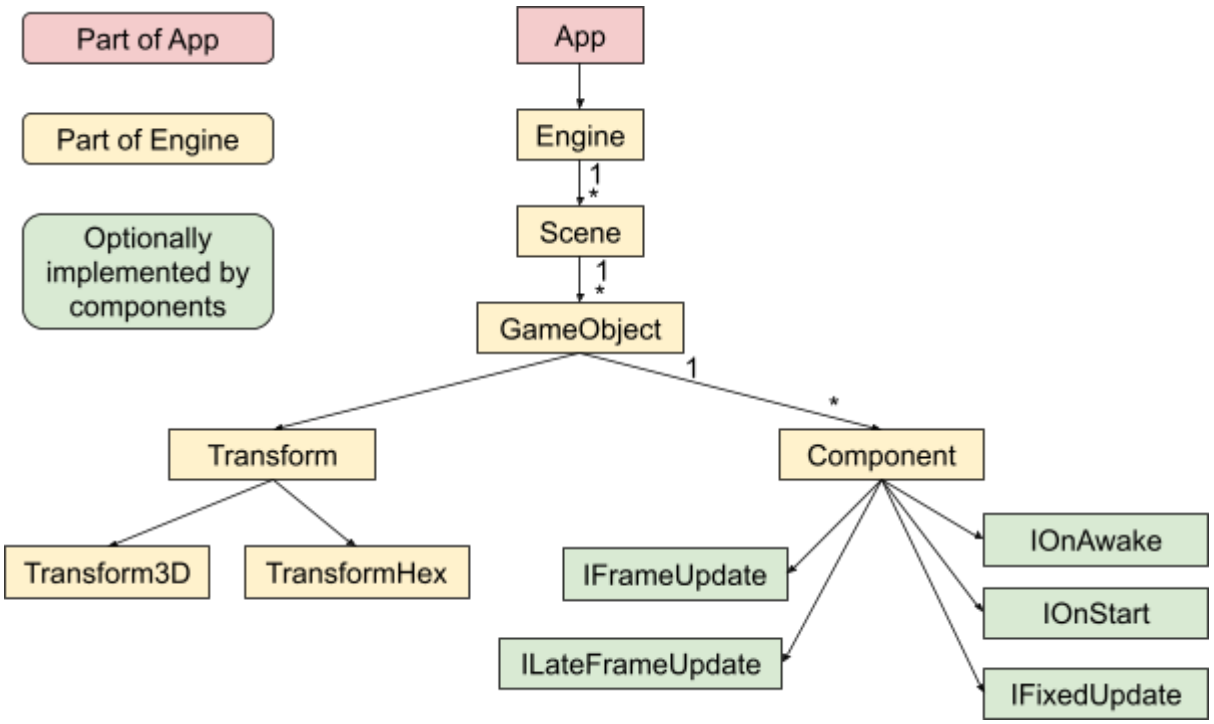
Audio		
1	The system shall allow for background music to be played.	M
2	1. The system shall allow for spot effects to be played.	M
	2. The system shall allow for multiple spot effects to be played simultaneously.	S

	3. The system shall play spot effects when key actions occur.	M
3	1. The system shall allow audio to be muted.	M
	2. The system shall allow for audio volume to be adjusted.	C
4	The system shall preserve audio options across multiple launches of the game.	W
5	The system shall fade volume by distance from the location of the spot effect.	W

Input		
1	The system shall accept keyboard input.	M
2	1. The system shall accept mouse button input.	M
	2. The system shall allow for mouse scrolling to be detected.	C
3	1. The system shall track mouse position.	M
	2. The system shall allow for mouse dragging to be detected.	C

3. Software Design

Our game uses a component architecture, inspired by Unity3D engine, where customizable components sit on individual game objects. In addition to components, game objects have relative transformation, and also contain child game objects, which creates world hierarchy. Finally, multiple root game objects are stored on a Scene, which is managed and updated by the underlying engine, which is initialized by the game app. We chose this architecture, because it allows for a high degree of customization for the environment, and game behaviour, in a way that is decoupled from the engine itself. This allows us to separate the game, and the engine into two halves. In addition, this architecture has proved to be very successful in multiple widely used game engines. We customized this architecture for the purposes of our game, to include multiple transformation classes which allows most of game logic to work in axial coordinates, simplifying the workflow while keeping the flexibility of 3D coordinates for other key components. As for networking, we employ a client-server based approach, which is described in detail later in this document. For clear separation of client, and server states, the engine loads duplicate scenes for each of them, and simulates them individually.



3.1 Division between engine and game

Engine components are somewhat special, in a sense that the engine knows about them, and uses them in more powerful ways by underlying engine systems. The Engine will hold the main loop whilst the game components will house the logic of the actual game. Game components are concerned with the game, and may use engine components to aid them, but the engine treats them equally.

3.2 Initialisation

To initialize the game, we first have to load initial *GameObject* templates and set up the initial scene. Once this is done the engine will be started and it will enter its main loop and invoke the game logic.

3.2.1 Constructing GameObjects

The GameObjects represent buildings, props and scenery. The *GameObject* by itself is a just container that has Components, while these Components implement actual functionality. We construct them from base templates in which *GameObjects* can be created, configured and stored with all of its components, properties and child *GameObjects* as reusable assets. These templates are particularly useful when multiple copies of the *GameObject* are required as it keeps all the copies automatically in sync. These templates can then be instantiated into the scenes at will. The templates and parts of game scenes are built in Blender, a free 3D modelling software, and then converted to glTF files, which is a royalty-free format for scenes, and models, created by Khronos group. They get loaded in as scenes, and then root objects can be used as templates by the game logic.

3.2.2 Constructing scenes

We have the main menu, and the game world as separate scenes. These scenes contain their entire world hierarchy and can be swapped at will. They are constructed in a hybrid approach - most 3D objects are set up inside Blender, while the UI and UI-interfacing 3D objects get set up programmatically. The game is able to load the scene using *Engine::loadScene*, or *Engine::activateScene* methods. The game will also want to set a particular scene as a presentation scene with *Engine::loadPresentationScene*.

3.3 Main loop

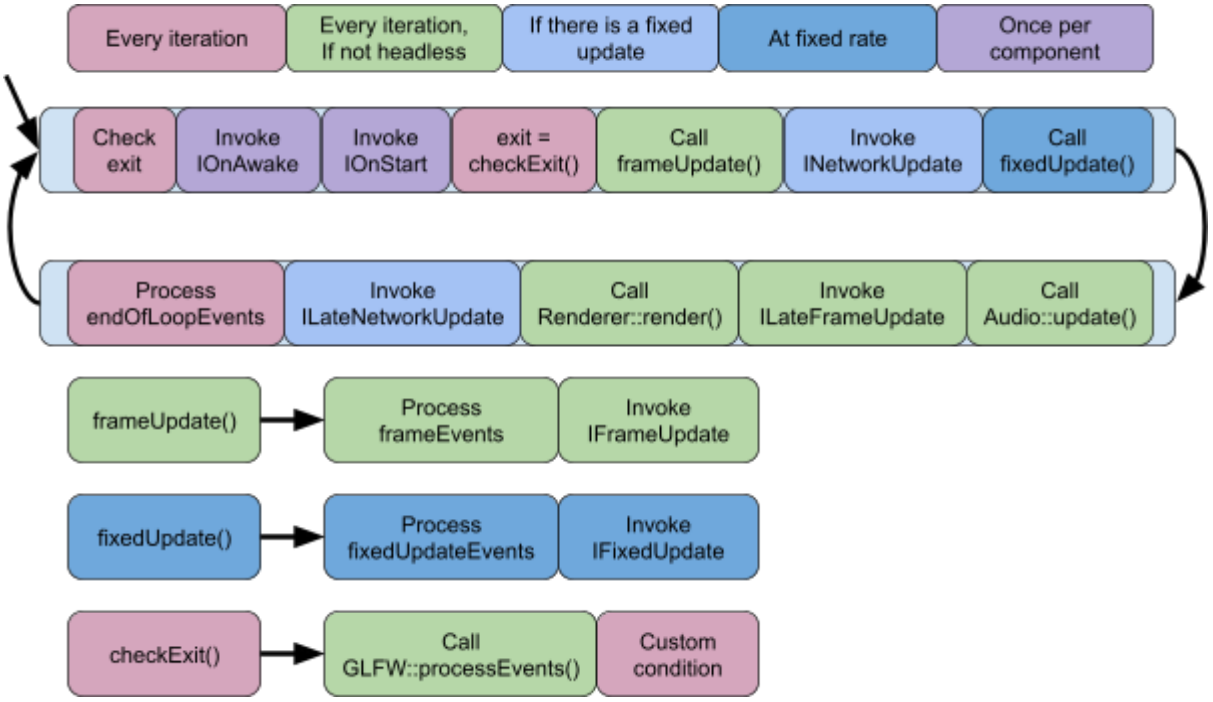
The main loop in the engine will check for any inputs from the user, then update all the components (with each component having its own sequence of actions to complete, at different stages of the main loop), then it will render the scene and finally check if the exit condition has been met. Multiple scenes are simulated this way, and the only difference is that there can only be one single ”presentation” scene. Only this scene’s components receive frame updates and get rendered. In addition, components can schedule custom events into various stages of the pipeline, on top of which a futures-like API is built upon.

Update stages are defined by components by implementing a number of custom interfaces:

IFrameUpdate - This interface will be invoked on the presentation scene every frame and contains any logic which is required to be updated every frame.

IFixedUpdate - This interface will be invoked N times per second (on average) and will contain any logic which needs to be updated every tick (invocation of the function). It is called on all scenes.
ILateFrameUpdate - This interface is essentially the same as *IFrameUpdate* but is invoked right before rendering (also on presentation scene).
INetworkUpdate - This interface is only used by network managers to synchronize state before *IFixedUpdate* is invoked.
ILateNetworkUpdate - This interface is used by network managers to send out updates after all component updates are performed.
IOnAwake - This interface is invoked once when a component appears in a scene and this method will let them set up any self-contained data needed by them.
IOnStart - This interface is invoked once when a component appears in a scene and will allow them to make sure references to other components are initialized.

Diagram below illustrates the main loop:



3.4 Component structure

A component is an object attached to a *GameObject* and implements functionality. This is done by extending Component, and implementing different interfaces mentioned before, which will specify what the component should do.

3.4.1 Core engine components

We used CRC (Class, Responsibilities, Collaborators) cards to describe what each component does, contains, and which components it collaborates with. For higher clarity and conciseness, we omitted listing non-component collaborators, private methods and members, repetitive helper methods, and miscellaneous methods that are used internally, but would not be helpful for a game. Here is an example of how the fields are laid out:

Class Name	
Attributes Methods	Collaborators

Each of the components listed below extend this base *component* class:

abstract Component	
Reference<Component> mReference GameObject mGameObject boolean mAwake boolean mEnabled boolean mStarted	

Transform is to store and manipulate the location, scale and rotation of the object as a matrix. Each transform uses a parent transform, that allows you to apply position, scale and rotation hierarchically.

abstract Transform	
abstract Matrix4fc getWorldMatrix() Matrix4fc getMatrixForChildren() Matrix4fc getInvWorldMatrix() Quaternionf getRotation() AxisAngle4f getRotationAngles() Vector3f getPosition() Vector3f getScale() abstract void setLocal3DTransformation() Vector3f transformDirection(Vector3f direction)	GameObject

Transform3D - transformation object that uses 3D location, and scale vectors, as well as rotation quaternions to compute the local matrix.
TransformHex - transformation object that uses simplified axial coordinates to represent location on a hexagonal grid and to compute the local matrix.

Transform3D extends Transform	
Vector3f mPosition Quaternionf mRotation Vector3f mScale	

TransformHex extends Transform	
Vector2f mPosition //axial float mRotation float mHeight Vector3f axialToCartesian(Vector2fc axial, float height) Vector3f roundCube(Vector3f cubePoint) Vector3f axialToCube(float q, float r) Vector2f cartesianToAxial(Vector3fc cartesian)	

The *Renderable* makes the object appear on the screen.

Renderable	
Mesh mMesh IMaterial mMaterial Vector3f mAABBMin Vector3f mAABBMax float getDepth(Vector3fc camPos) boolean frustumCull(FrustumIntersection intersection)	Transform

The *Camera* is a collection of transformation properties used by the renderer.

Camera	
Projection mProjection = Perspective | Orthographic float mFov float mOrthographicSize float mNearPlane float mFarPlane float mAspectRatio Vector3f mViewDirection Vector3f screenToWorldDir(float x, float y) Vector3f screenToPlane(Transform transform, float height, float x, float y) Matrix4fc getView()	Transform

Light is a collection of properties representing directional lights to be passed to shaders.

Light	
LightType mLightType float mIntensity Vector3f mColour	Transform

The *AudioSource* is a component that acts as a source from where audio plays.

AudioSource	
Sound mSound Source mSource float mVolume float mPitch float mRadius float mTimeLeft int mLooping void attachSource(Source source) void detachSource() void playSound(String name) void playSound(Resource<Sound> sound)	DataLinePool

The *AudioListener* acts as a marker for the audio system for where to listen for sound from.

AudioListener	

The *NetworkObject* acts as a bridge between the server and client for each *GameObject*. It communicates the changes to a *GameObject* between a server and client.

NetworkObject	
int mNetworkObjectId int mOwnerId NetworkManager mNetworkManager	NetworkableComponent NetworkManager

NetworkableComponent acts as a marker and allows variables, requests, and events to be synchronized across the network.

abstract NetworkableComponent	
NetworkObject mNetworkObject void onNetworkInitialise() void onConnectedSyncvars() void beforeNetSerialize() void afterNetUpdate() void onOwnerIdChange(int newId)	ISyncVar NetworkObject

NetworkManager is a common component that will either house a server, or a client network manager, and provide a common interface between them.

NetworkManager	
TemplateManager mSpawnableTemplates Scene mGameScene ISceneBuilder mGameSceneBuilder ClientNetworkManager mClientManager ServerNetworkManager mServerManager boolean isServer() boolean isClient() void createClient(String ip, int port) void createServer(int port) Integer findTemplateByName(String name) float getServerTime() Stream<NetworkObject> getNetworkObject() Stream<NetworkObject> getObjectsOwnedBy(int ownerId) NetworkObject getObjectById() SingletonStore getIdSingletons(int ownerId) void closeInstance()	ClientNetworkManager ServerNetworkManager

ServerNetworkManager provides server-specific methods and variables

ServerNetworkManager	
ServerGameState mGameState Reference<NetworkObject> spawnNetworkObject(int templateId) Collection<ServerClient> getClients() void destroy() SingletonStore getIdSingletons(int ownerId)	ISyncVar NetworkObject

ClientNetworkManager provides client-specific methods and variables

ClientNetworkManager	
ConnectionState mConnectionState int mNetId float mServerTime void sendToServer(byte[] message) DataOutputStream getDataOut() Stream<Reference<NetworkObject>> getNetworkObjects() void disconnect() SingletonStore getIdSingletons(int ownerId)	ISyncVar NetworkObject

3.4.2 Communication between components

The primary way to communicate between classes is to register a singleton on the active scene. A component, for example, GameLogic, would call *Scene.getActiveScene().registerSingleton(this)* inside its *onAwake* method. Then, any other component is able to retrieve the singleton after onAwake by calling *Scene.getActiveScene().getSingleton(GameLogic.class)*, which is then callable and points to one singleton GameLogic instance. Alternative way, used between components on the same game object, is to call *getGameObject().getComponent(Type.class)* method. This way is used in close object relationships, inside many scenarios like visual effects, object actions, etc. It is preferred to call this method only once, since it has O(N) complexity, where N is the number of components on the game object.

3.5 Core systems

In this section, we will describe some of the core systems of our engine, how they are built, and how they drive our game.

3.5.1 Audio system

The audio system makes use of the OpenAL bindings in the lwjgl library for playing sounds. The reason we chose to use OpenAL over something like Java Sound was because it is more efficient, and allows for positional audio without the need to do any of the complex math ourselves. The audio system is initialised when the game is started by opening the default device, creating the OpenAL context and then generating up to 32 sources. The number of sources can be changed to any value and as long as one source is generated, audio will work, although only one sound would be playable at any one time. Once the *AudioManager* is initialised, *AudioSources* can be added to any game scene and to play a sound *AudioSource::playSound* is called, either with the name of the sound file, or with a *Resource<Sound>* object. Audio is not played until the *AudioSource* has an OpenAL source attached to it. When a source is attached the *AudioSource* components set the position of the OpenAL source every *fixedUpdate* so that audio is always heard from the correct direction relative to the *AudioListener*.

The *AudioManager* class handles the distribution of OpenAL sources to *AudioSource* components and is also responsible for updating the position and rotation of the OpenAL listener. This is all done through the *AudioManager::update* method which is called every single frame in *lateFrameUpdate*. The update method starts by updating the position and rotation of the OpenAL listener, before filtering the list of all *AudioSources* in the current scene to keep only those that are valid. The criterion for an *AudioSource* to be invalid and thus removed from the list are as follows:

-The *AudioSource* has no Sound to play
-The *AudioSource* has already finished playing it’s sound (mTimeLeft < 0)
-The distance from the *AudioListener* to the *AudioSource* is greater than the source's radius.

Once the list of *AudioSources* has been filtered, the *AudioManager* attaches sources to all of those in the list, up until the number of OpenAL sources has been reached. Currently, the only audio format that is supported is .wav, however the system is designed in such a way that adding new formats is incredibly easy, and would not require any refactoring.

3.5.2 Input system

When designing the input system, we wanted to provide a simple interface that delivers a logical distinction between input provided and the events they trigger. This was implemented via *Bindings* and *Actions*, respectively.

Each *Action* represents an event. Every *Action* provides a small set of functions that can be easily used within the game to determine the flow of logic, based on the fact that the event may be triggered. To implement the actions needed by the game, it is a simple case of extending the abstract *Actions* class, as done in *GameActions*.

Action	
boolean isActivated() boolean isJustActivated() boolean isJustDeactivated() float getActivationTime()	

Bindings are required to map keyboard and mouse buttons to each game *Action*. To implement bindings in the game, we simply extend our abstract *Bindings* class, as done in *GameBindings*. This class allows for many buttons to map to many *Actions*. For example, this allows us to bind both the 'W' and 'Up Arrow' keyboard buttons to a single 'UP' Action.

To utilise these custom *Actions* and *bindings*, all we need to do is provide the *GameBindings* to the input system. The input system- utilising GLFW's keyboard and mouse polling to detect button presses- then automatically handles button presses and the activation of relevant *Actions*, as defined by the provided bindings.

This system is designed to allow for greater flexibility and extensibility, due to the distinction between hardware triggers and logical events, with it even being possible to programmatically change bindings while the game is running.

The input system also provides additional functionality outside of basic keyboard and mouse button presses. Cursor acts as an interface for input related to mouse movement, providing the cursor position scaled to the window size, and allowing mouse dragging to be detected. Scroll provides an easy way to get input related to the mouse scroll wheel, from precise scroll values to giving the ability to use Scroll.UP and Scroll.DOWN as special buttons (that can be bound to *Actions*).

Overall, the input system is designed to be simple to use while providing extensibility, due to the distinction between hardware triggers and logical events. Furthermore, due to this separation, it is possible to programmatically change bindings while the game is running (which is currently not used- but provides a future method to implement custom key bindings per user).

3.5.3 Rendering system

For rendering we wanted flexibility, and high performance, thus we chose Vulkan API. From the game, all rendering complexity is abstracted away. An object with a renderable assigned to it will be rendered.

Each frame, the engine collects a list of objects to be render, the main camera to use, and a list of lights that are in the scene, and then call the render method on *Renderer*:

Renderer	
void render(Camera camera, List<Renderable> objects, List<Light> lights) void onResize() void setSettings(RendererSettings newSettings) void free()	

Upon rendering a frame, the *Renderer* will take a list of objects, group them into instanced batches, update any buffers on the GPU, and invoke Vulkan draw commands. The draws are instanced, meaning a single draw call may be used for multiple objects. Generally, all opaque objects that have the same mesh, shader set and textures can be batched. All transparent objects are sorted front to back to achieve a consistent transparency look.

Only when creating custom materials, some knowledge of Vulkan is needed, due to the *IMaterial* interface:

IMaterial	
ShaderSet getShaderSet() int writeVertexInstanceData(int offset, ByteBuffer buffer, Matrix4fc matrix, List<Light> lights) SampledTexture[] getFragmentTextures()	

Here, a material is responsible for writing its properties into a vertex instance buffer, which then gets passed into a vertex shader, specified inside the *ShaderSet*. Additionally, *ShaderSet* also allows a material to specify geometry and fragment shaders, among other properties, such as data layout.

Luckily, this is rarely needed, because the standard *PBRMaterial* can be used for almost any scenario. It is a physically based material that has customizable albedo, normal, metalness, and roughness map textures and values. It supports transparency and alpha clipping. Shader variants are handled behind the scenes, where unused features get disabled. All objects imported from Blender have this material applied to them, and the rendered result is almost identical between the game and Blender, which allows artists to preview assets inside the 3D software, before importing them into the game.

The only case where we needed a customized 3D object material was hexagon tiles. There, we wanted to add a custom highlight to map tiles, where edges of the hexagon are more highlighted than the center. Even so, we just inherited from *PBRMaterial*, and added a few additional parameters to the shader.

Overall, our rendering system is extremely powerful, allows for a high degree of customizability, and maintains sufficiently high performance.

3.5.4 Network architecture

Network architecture is split into two parts - low level transport, and high level API. Low level transport is concerned with one thing, and one thing only - reliably passing byte messages between the client and the server.

For low level API, we have considered using either TCP, or UDP protocol. UDP is typically used for games, because exact data reliability is not a must-have, and can always be corrected over time. It also typically has lower latency, and avoids congestion problems. Since our game is not latency sensitive and TCP has numerous advantages, such as reliability out of the box, message ordering, and overall a much simpler way to do network communication, we chose not to write a UDP layer, and instead used TCP which gave us more time to focus on other parts of the engine.

For high level API we employ a client-server architecture, where the server has full authority, while the client knows only what it needs to know, and simply sends commands. This increases security, and indirectly - coding standards. Each networkable *GameObject* contains a *NetworkObject* component, which allows networkable components to synchronize state, send commands from clients, and receive events from the server. Such components extend the *NetworkableComponent* class.

State is synchronized automatically by *NetworkObject* through the use of *SyncVars*. They are classes that implement *ISyncVar* and *INetSerializable* interfaces:

ISyncVar extends INetSerializable	
boolean isDirty(int clientId) default void resetDirtyFlag() default void resetDirtyFlag(int clientId) default void setClientDirty(int clientId)	

INetSerializable	
void serialize(DataOutputStream stream, int clientId) void deserialize(DataInputStream stream)	

There are several *SyncVar* types to be implemented for primitive types: *SyncShort, SyncInt, SyncLong, SyncBool, SyncFloat.* Other *SyncVar* types are implemented for other types: *SyncString, SyncVector3*.

Each *NetworkableComponent* stores a list of *ISyncVar* fields internally, and *NetworkObject* calls *sendUpdate* on the server, whenever the state needs to be serialized. Variables that have not changed, can avoid synchronization, thus, *isDirty* function is called beforehand. *isDirty* values are stored on a bitfield, and only the changed variables get serialized. This bitfield is then added to the beginning of the update message. The client then receives the serialized bytes, and *NetworkObject::updateFromBytes* is invoked. Deserialization occurs in the same order. First, the client reads the bitfield and calls deserialize individually on each *SyncVar* that has changed.

These networkable *GameObjects* are always created by the server, from a list of networkable Templates, which are loaded from a glTF file. Networkable objects always have a *NetworkObject*

component attached. Upon creation, a unique *networkObjectId* is created, and the spawn event is broadcast to the clients, who should see the spawn happen.

Additional features in our networking code are server events, and client requests. They allow the game to communicate custom instructions between clients and servers. *ClientRequest*, as the name implies, allows clients to send requests to the server. Such requests are always authenticated, meaning, only requests coming from the owner of a particular object are processed. Each networkable component can define a list of client requests, and must be initialised inside the *onNetworkInitialise* method.

ServerEvent is similar to *ClientRequest*, the main difference being communication flows from the server to the clients. Also, there is additional information attached to the event, such as target recipients of it, and event persistence. It is possible to send the event privately, just to the object owner, to all active clients (clients on whom the object is visible, more on that in the next paragraph), or to all clients in the game.

We originally envisioned clients receiving updates only of the objects they were supposed to see. It could automatically function as a fog of war, even if there were no visual cues implemented. This is often referred to as dormancy, and is the primary way to render visibility hacks worthless. Unfortunately, due to time constraints we did not fully implement this system, and we only have it for hexagon map synchronization, but the network code was designed with it in mind, and would not require refactoring to add to the existing code. The rest of this paragraph describes the changes needed for dormancy to function. Some events are one off (like battle effects), but others are persistent (like object creation, and destruction). Persistent events need to be queued for each client, and sent when they become visible. *GameObjects* can become invisible at any point. In these scenarios, the client gets told that the object has become dormant. When dormant, no updates get sent about the object, and upon the object becoming active, full *SyncVar* state is synchronized (for simplicity), or a destruction event is sent. If an object gets created, and destroyed all the while the client is not sent the updates, the server should just remove the spawn event and act as if the object never existed. Just like dormancy, persistence was implemented selectively (only for object creation). However, appropriate flags and controls have been implemented in *ServerEvents*, so all that is missing here is actual feature implementation.

Players are connected through the use of a lobby system. On local networks, UDP discovery is used to find the host. Outside LAN, hosts attempt to automatically port-forward the game through UPnP, and then register themselves on a serverless API. Clients can then get open lobbies, and connect. However, we encountered an unexpected issue with Java HTTPS requests simply failing on approximately 1 out of 5 machines with no clear reason. Thus, as a workaround we ended up hosting the lobby on one of our personal cloud servers.

Overall, while not every wanted feature was implemented, our network architecture already allows for secure, reliable, and compact communication between clients and servers. With, on average, 3 kilobytes per second of bandwidth used on the clients, it is possible to play the game on highly restrictive network conditions, such as 2G and 3G cellular connections.

3.6 Client-side representation of the board

Below we list some of the core game components. This list is not meant to be exhaustive, and only highlights the most important parts of the game, and how they interact with the engine components. Like the engine components, some helper methods are omitted, because they are very similar.

Player is the base class that allows actions in the game. It does not invoke any actions by itself, however. Instead, it is composed and controlled by 2 classes - *HumanPlayer* and *AiPlayer*. This allows us to share code, provide the same level of functionality for both regular players and the AI, as well as easier testing (clear, openly accessible ways to invoke requests).

Player extends NetworkableComponent implements IOnStart, IFixedUpdate	
Map<HexagonTile, Building> mOwnedBuildings Reference<Building> mCapital SyncInt mTokens SyncVector3 mPlayerColour SyncBool mOwnsCapital SyncFloat mLastAttack Reference<HexagonMap> mMap Reference<GameState> mGameState Reference<MapEffects> mMapEffects void buildRequest(BuildData data) (ClientRequest) void attackEffect(AttackData data) (ServerEvent) void attackRequest(AttackData data) (ClientRequest) void statRequest(StatData data) (ClientRequest) void sellRequest(SellData data) (ClientRequest) boolean buildCheck(HexagonTile tile, int buyPrice) boolean attackCheck(Building attacker, Building defender) boolean sellCheck(Building building) boolean statCheck(Building building, StatType statType) void addOwnership(Building building) boolean removeOwnership(Building building) int getTileViewability(HexagonTile tile) Stream<HexagonTile> getViewableTiles() List<Reference<Building>> getOwnedBuildings() boolean hasLost() boolean gameEnd()	Building HexagonMap GameState MapEffects HexagonTile

Buildings get created by the players upon build requests, and they contain stats, control visual effects.

Building extends NetworkableComponent implements IOnAwake, IOnStart, IFrameUpdate	
SyncStat mAttack SyncStat mDefence SyncStat mTokenGeneration SyncStat mViewDistance SyncStat mBuildDistance SyncStat mClaimDistance SyncBool mIsCapital SyncFloat mActionLockTime Set<HexagonTile> mClaimedTiles void onClaimTile(HexagonTile tile) void attackEffect(Building defender, float attackTime) boolean attack(Building opponent) Set<Building> getAttackableBuildings() Building getRandomAttackableBuilding() List<HexagonTile> getAttackableTiles() boolean isActionLocked() boolean isBuildingAttackable(Building target) Set<HexagonTile> getBuildableTiles() HexagonTile getTile() int getAttackCost() void remove()	HexagonMap Player HexagonTile

HexagonMap is a class that generates and stores the game map. It is randomly generated on the server, and pre-filled with fog tiles on the clients.

HexagonMap extends NetworkableComponent implements IOnAwake	
HexagonTileStore mTiles HexagonTile cursorToTile() Stream<HexagonTile> getAllTiles() void floodFill(Deque<HexagonTile> tiles, IFloodFillVisitor visitor)	HexagonTile

HexagonTileStore is a *SyncVar* container that stores hexagon tiles, and performs culled map updates to the clients

HexagonTileStore implements ISyncVar	
Stream<HexagonTile> getAllTiles() HexagonTile getTile(int q, int r)	HexagonTile HexagonMap

HexagonTile is not a component, it's a dedicated syncable object that is stored on *HexagonTileStore*:

HexagonTile implements INetSerializable	
TileType type = Land | Water | Mountain | Fog int mQ int mR float mHeight Reference<Building> mBuilding Reference<Building> mClaimedBy boolean setClaimedBy(Building building) void removeClaim() Player getClaimant() void setBuilding(Building building) boolean isBuildable(Player player) float getSurfaceHeight()	Building

3.7 AI Players

Our AI component of the game used the server to control other opponents on the map who were not controlled by a user. This meant that the AI players should have access to the same controls which the user is able to but still be able to make intelligent moves or moves which make it interesting for the user. To do this we created two different kinds of AI players. One which we called a *ProbabilisticAi* and another called *AimerAi*. *ProbabilisticAi* is a subclass of *AiPlayer* while *AimerAi* is a subclass of *ProbabilisticAi.*

The abstract class *AiPlayer* (which extends *Component*) contains the methods which both AI Players need access to and contains the *fixedUpdate* for the AI. In *fixedUpdate* it will firstly check whether it can make a move. This is done by *shouldPlayGame* which will check to see how long it has been since the last move and whether it has been longer than a random number (which is between a lower bound and upper bound). Then when it plays it will run the abstract method *simulateInput()* which will contain the code which is used by different Ai Players.

ProbabilisticAi players use probability to decide on the next move to take. Thus they have different chances to play different moves like attacking or building. For each of the main 4 actions (building, attacking. selling, upgrading) they choose a valid move to do with uniform probability. *ProbabilisticAi* are not used by themselves but rather in conjunction with *AimerAi* as they would not be very interesting players to play against.

AimerAI are AI players who aim for capitals and buildings when they are in the viewable tiles for that player. It also has a small chance to immediately attack the capital (which gets larger throughout the game) even though it cannot see it. To aim for a building or capital the *AimerAi* uses the A* algorithm to find the shortest route to the building to attack. When the *AimerAi* is not aiming for a building (because it cannot see any) or with a small chance when it is aiming for a building, it will fall back to playing as a *ProbabilisticAi* player. This allows the AI to still upgrade and build thus strengthening its position.

All the values which decide probabilities, or length of time to wait between moves etc, are stored in the configuration file which allows them to be changed on the fly. This also allows it to be easier to add in new AI without having to change the codebase as well.

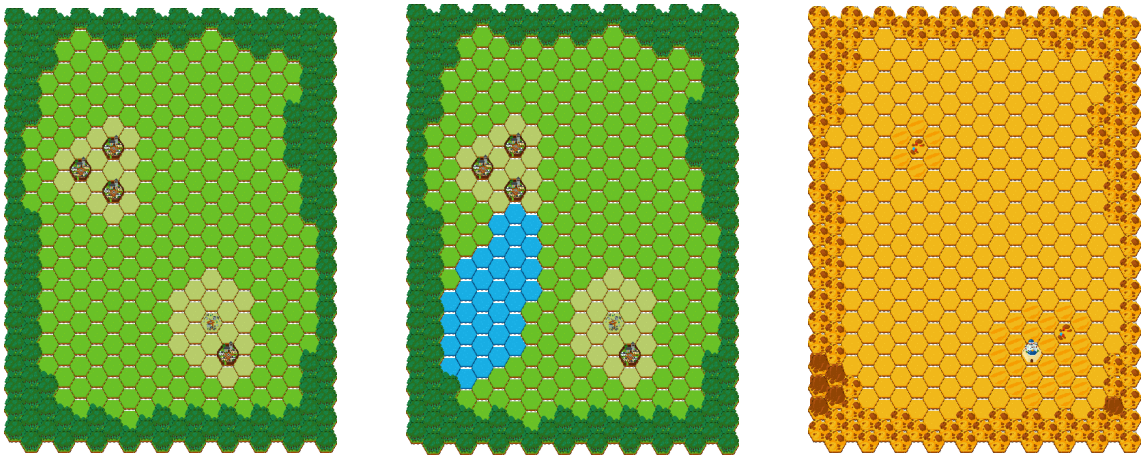
4. Interface Design

In a strategy game it is important for the correct data/screens to be shown to the user so they can only access the parts of the game necessary to the current game state. It is also important that the UI is simple and can be learnt by playing, with similar actions and information being displayed in the same position and the same graphics.

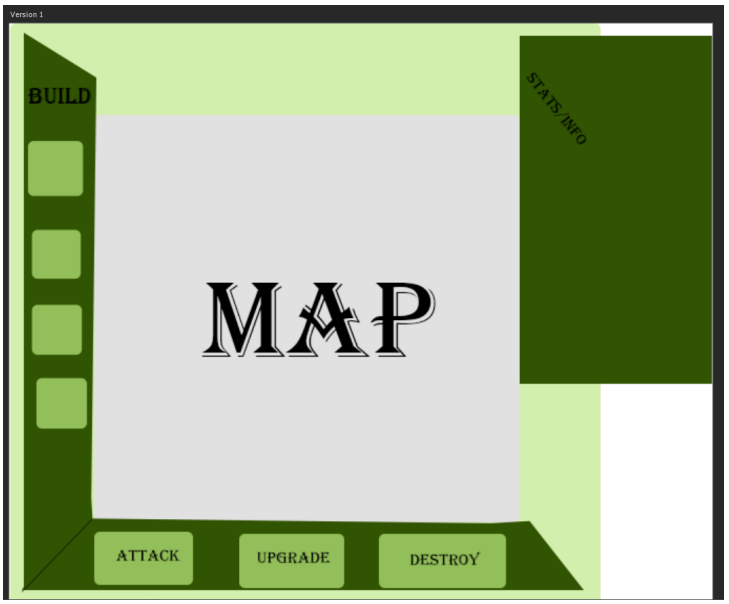
Our game has a basic theme surrounding hexagon's and flat UI, the blend between the two helps to separate the gameplay and the UI used for actions.

4.1 Initial Ideas

We wanted the map to consist of hexagons and have different tile types depending on its state; buildable, hidden or out of bounds. Below are some first ideas of this.

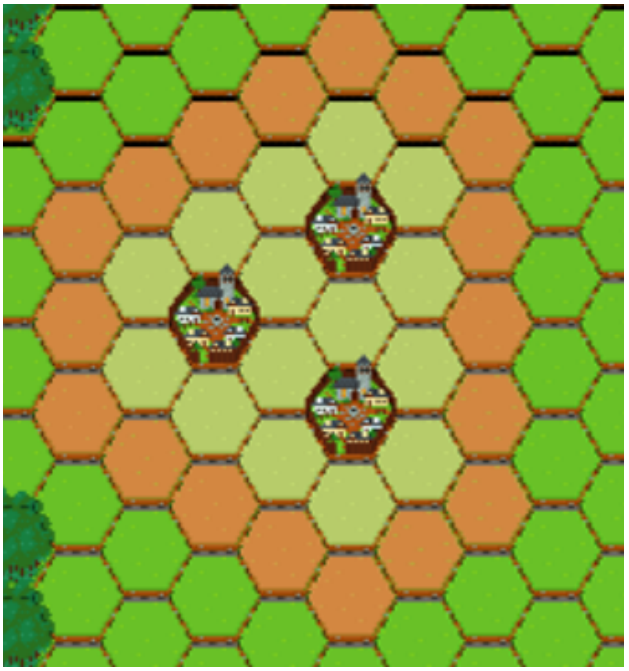


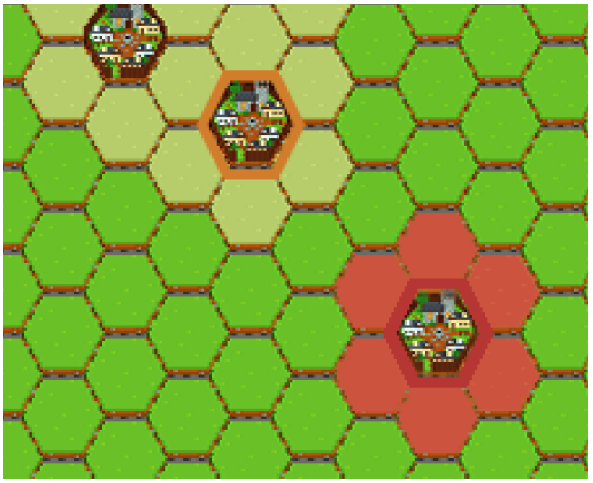
As for the layout of the UI we decided we want to keep them mostly separate so that the user can distinguish between the interactive area and the playable game areas. Shown to the right was the first layout we created, we found that the separation of the sections worked well but that it was too invasive as it restricted the playable area of the user. To fix this we moved everything into the left pane and expanded the playable area. Any small features such as the zoom bar will be positioned over the playable area as it doesn't fit with what will be shown in the pane. Below is the adaptation we came up with, the right image shows the screen after an interaction of the user. This design keeps all the UI changes separate and non-invasive.



4.1.1 Initial Playable Area Visible Changes

We wanted to utilise the map tiles as much as possible to represent the game state whilst also being intuitive to the user. An idea we had was to highlight the tiles surrounding the buildings to indicate what you can or cannot do at the current state. To the left is shown what a claimed (light green) and a buildable (orange) tile could look like. To the right is what launching an attack could look like with red tiles being attackable and the surrounded tile being the building instigating the attack.





4.2 Overview

After deciding on these initial concepts, as described above, we could implement our design in the game.

4.2.1 The Main Menu

When you first open the game you are presented with a few options, one to modify the settings and the other to start the game, which you will then decide if you want to Host a Game or join a pre-existing public/private game.

4.2.1.1 Host Screen

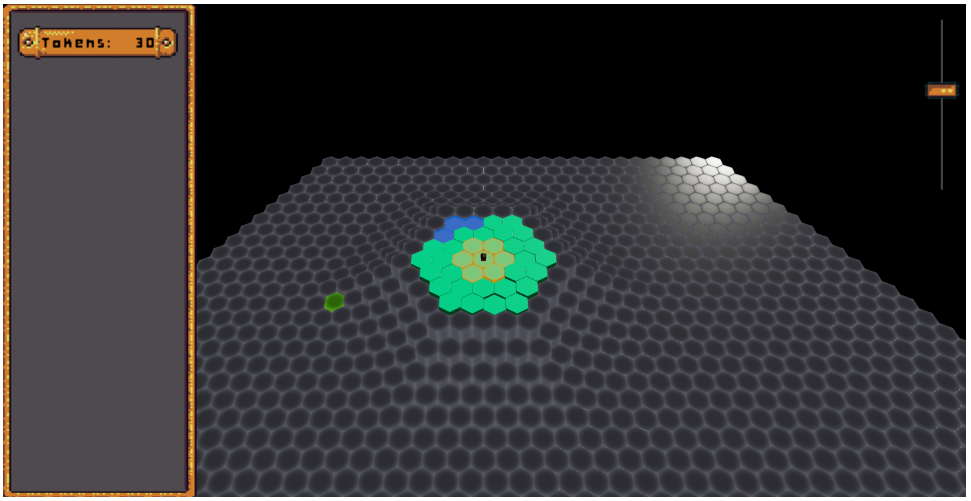
If a player decides to start the game as a Host, they will be directed to a Lobby screen where they can then start the game on all the clients at once.

Once gameplay has started the host is turned into a normal player and can play the game. Each player is given a highlight and this can be seen around their owned buildings, each building type also has a different model to reflect its highest stats, and also dynamic props surrounding it reflecting the individual stat levels.

4.2.2 Player Screen

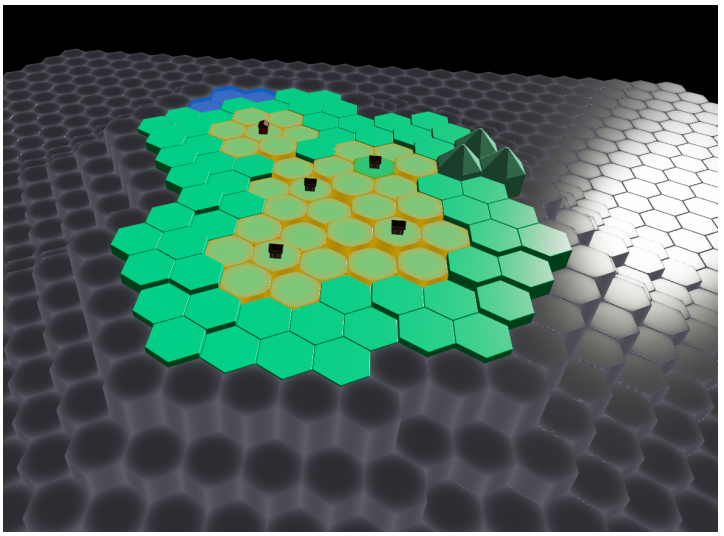
When a player starts they can either join a local lobby or a public one by entering in the game code. Once joined, the host will start the game and the player will be told this via visual prompts.

Once in the game their camera is moved to their main building, the capital building. They can see their claimed tiles highlighted in their player colour and nothing else as the rest of the map is undiscovered.



When the player builds a new building they can uncover more of the map and locate enemy buildings for them to take over. An enemy building is shown with a different colour highlight. When the player loses/wins a prompt will be displayed allowing them to either exit the game or to move around the map and watch the rest of the game. UI is disabled during this as they cannot play.

Areas where they cannot see are areas they have not discovered yet.



4.2.3 Building

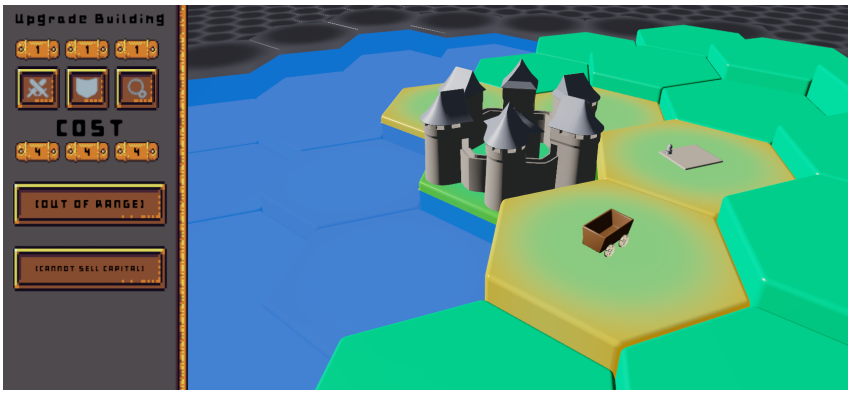
When a player selects a tile that is within their view, the tiles that can be built upon are highlighted. In order to build the player must first select a buildable tile, then the buildable options are displayed in the menu pane.

In the pane, possible buildings that can be built are shown with the number of stars indicating a higher level building. They are arranged into columns of similar type i.e. attack, defending and generation orientated buildings.

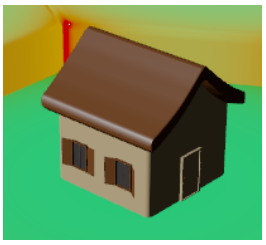
The selected buildings' statistics are also shown above along with its name. If the player has enough tokens to build then a build button is displayed, otherwise a warning saying they don't have enough tokens is displayed and the cost is highlighted red.

4.2.4 Upgrading

If the player selects their own building they will be presented with options to upgrade its statistics. The current level and cost are shown in the pane. Each button corresponds to its statistical level. E.g.



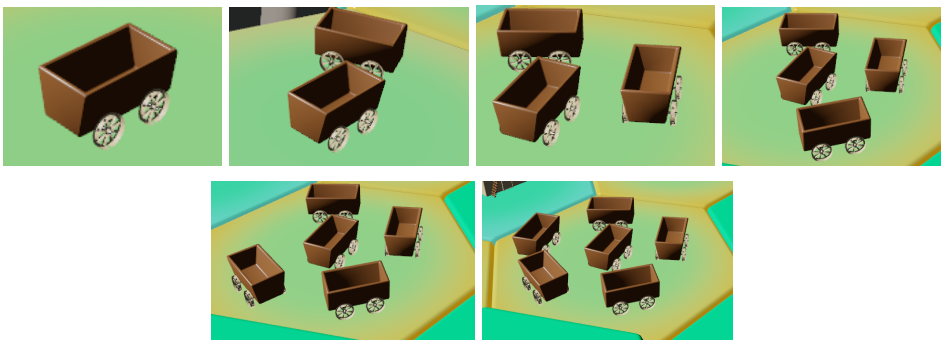
The first button is the attack upgrade. Each upgrade to a building can change its appearance, it will change its surrounding props but also the main building's mesh. Shown below are the models for the max building in each corresponding statistic.



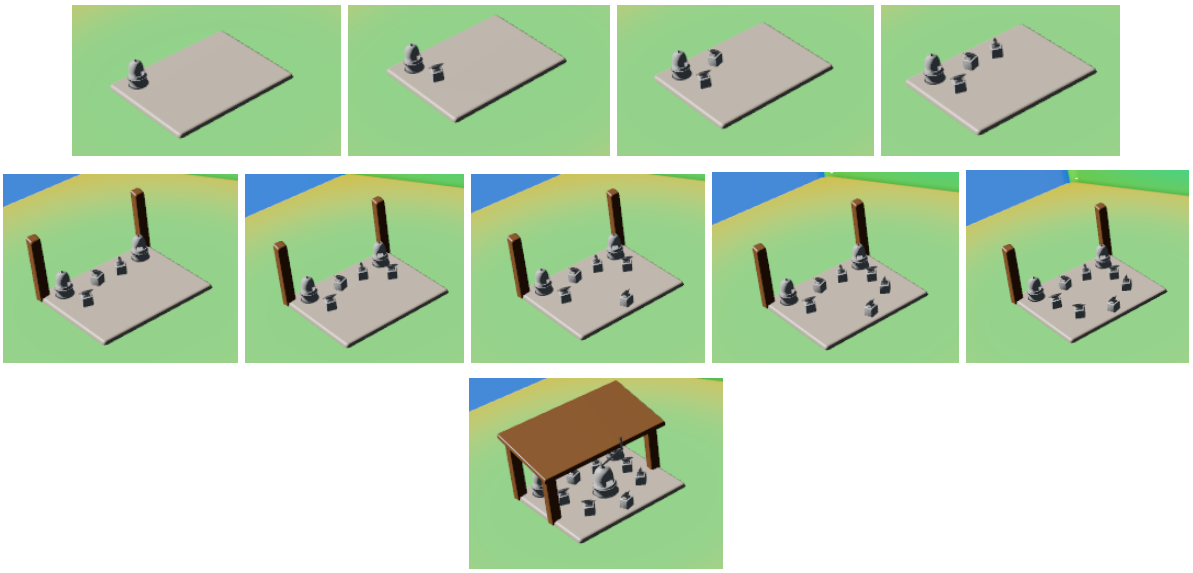
4.2.4.1 Surrounding Props

Each prop will change depending on its current level, this will be shown around the building and will ‘grow/build’ itself better when its level increases. This allows users to gain visual feedback about the level of a building.

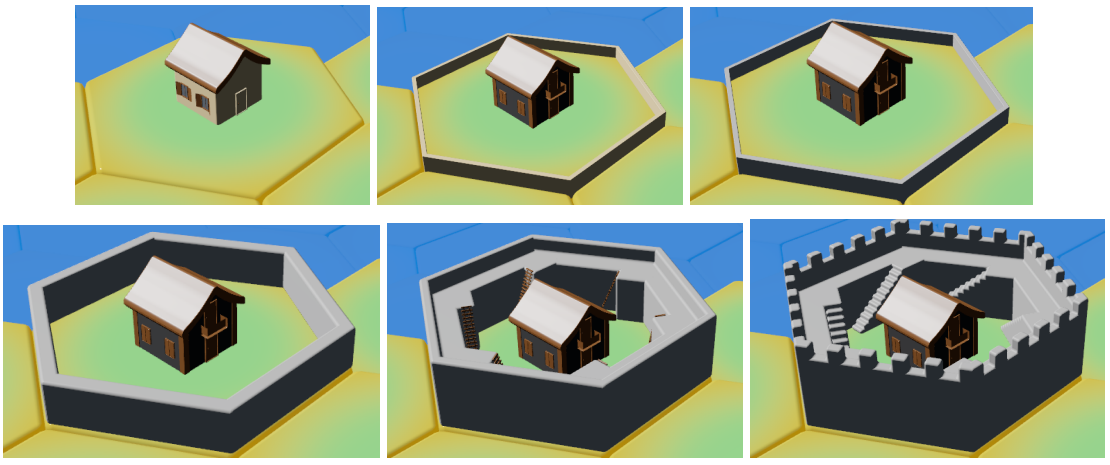
4.2.4.2 Attacking Prop Stages



4.2.4.3 Token Generation Prop Stages

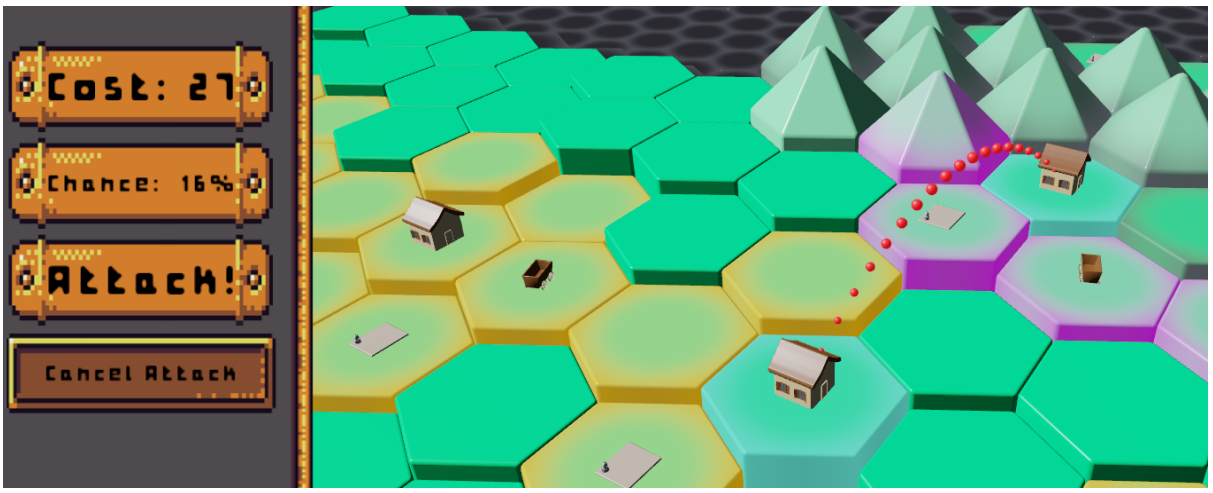


4.2.4.4 Defence Prop Stages

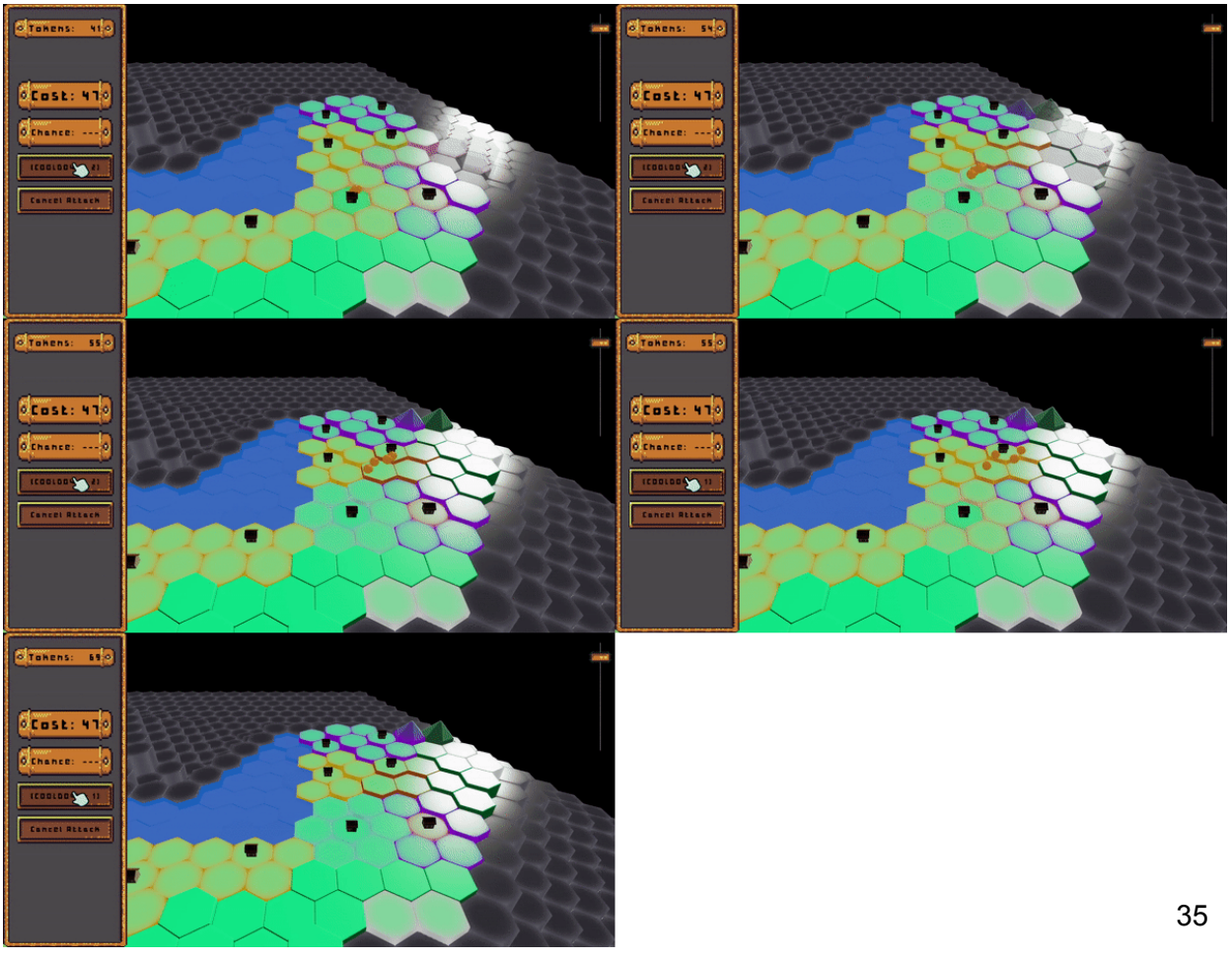


4.2.5 Attacking

When a player would like to attack another building, the possible attacking buildings are highlighted. The player is then prompted to select a building in which to attack, when they can then launch. Alongside this information is the cost to launch the attack in addition to the probability of success.

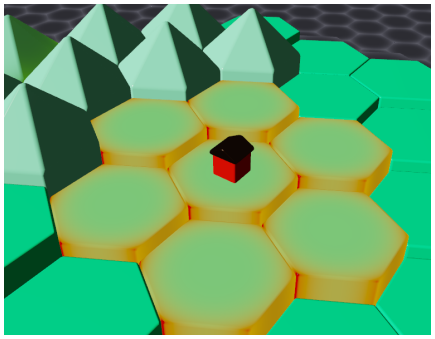


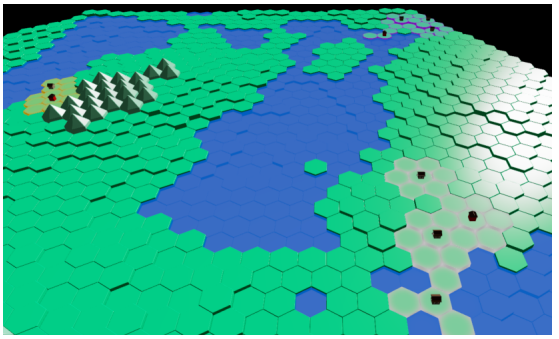
To indicate an attack being launched, projectiles are shown and the tile surrounding the buildings involved in the attack pulsate. The number of projectiles shown are dependent on the attack level. Shown below are a collection of frames showing the projectile.



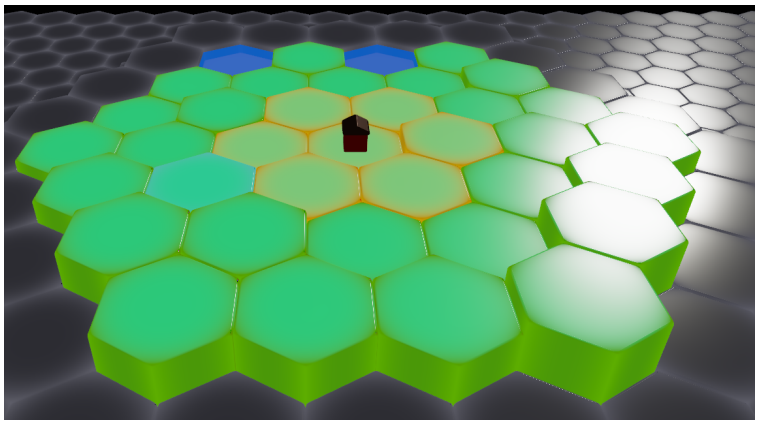
4.2.6 Tile Highlighting

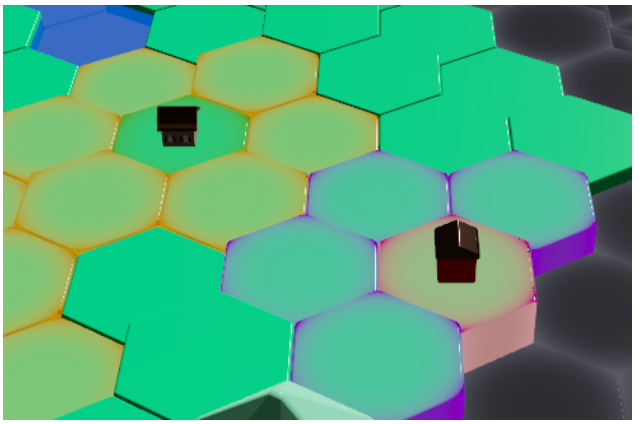
Throughout the gameplay, the tile highlight is used to represent the actions a player can take and who owns what tiles. Shown in the first photo is the highlighting for an owned tile, in this case orange is used to indicate our players owned tiles. In the second image we can see other players with different tile highlights indicating difference of ownership and hence a different player.





The third photo shows one, the selected tile from the user in light blue - this is different to any other tile so will stand out. And secondly in green the buildable tiles, in this photo the player is about to build a tile.

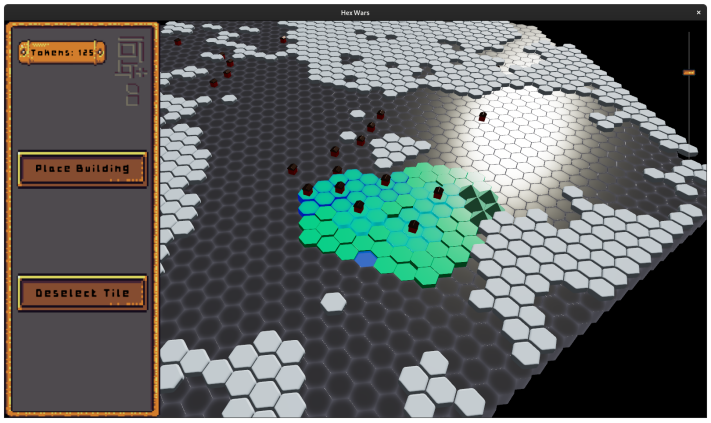


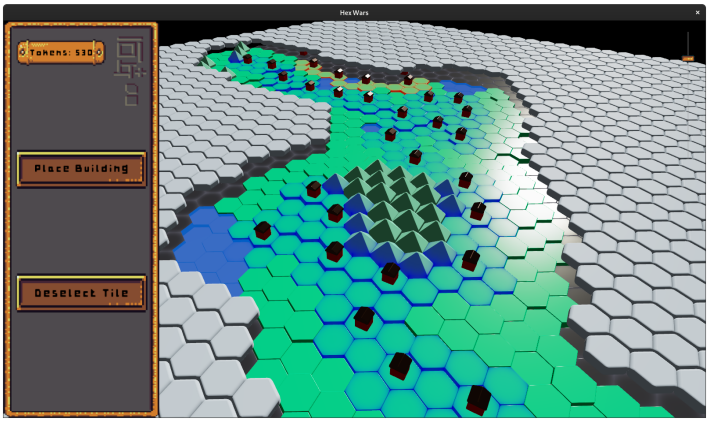


And finally in the last image, the player is about to attack a tile. The tiles which can be attacked are highlighted in red and the instigating tile is green as it is currently selected.

4.2.7 Fog Of War

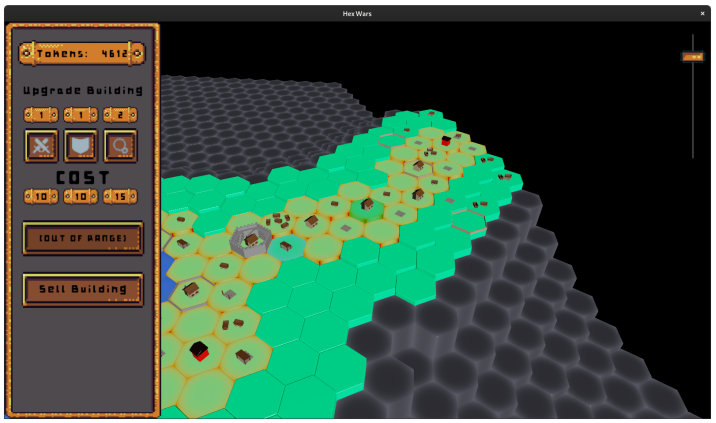
We went through two ways of hiding unexplored land from the user. One more realistic, one more abstract.

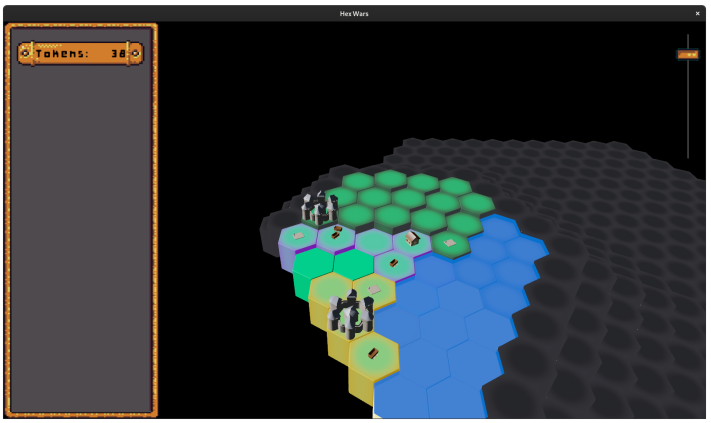




Initially we had a randomly generated ‘Cloudscape’ which would remove itself once land had been discovered. Although this looked good and worked very well, it was not very clear what the clouds were for or if they were event clouds. Another issue we ran into is that it wasn’t clear what it meant by an area of land to be discovered but also hidden (when an enemy has fought you out of the area). This was initially represented with no cloud cover but the land being visible. We also experimented with the clouds growing back but the land still being visible, this made it clearer what the clouds were but as a result made it even more confusing what was discovered and what was not.

The second implementation of the fog was by introducing tile heights, the discovered land being both non-black tiles and also being physically higher than the un-discovered. To represent discovered land that was lost to an enemy, we simply put a dark grey overlay onto the tile whilst keeping its height. This was distinguishable from other players as all players have a primary/secondary colour highlight.





We ended up using the second implementation as it was clearer and less intrusive. In a later development of the game we could perhaps introduce the clouds as a purely immersive feature, with them moving across the land in tune with the Day/Night cycles.

4.3 Design Choices

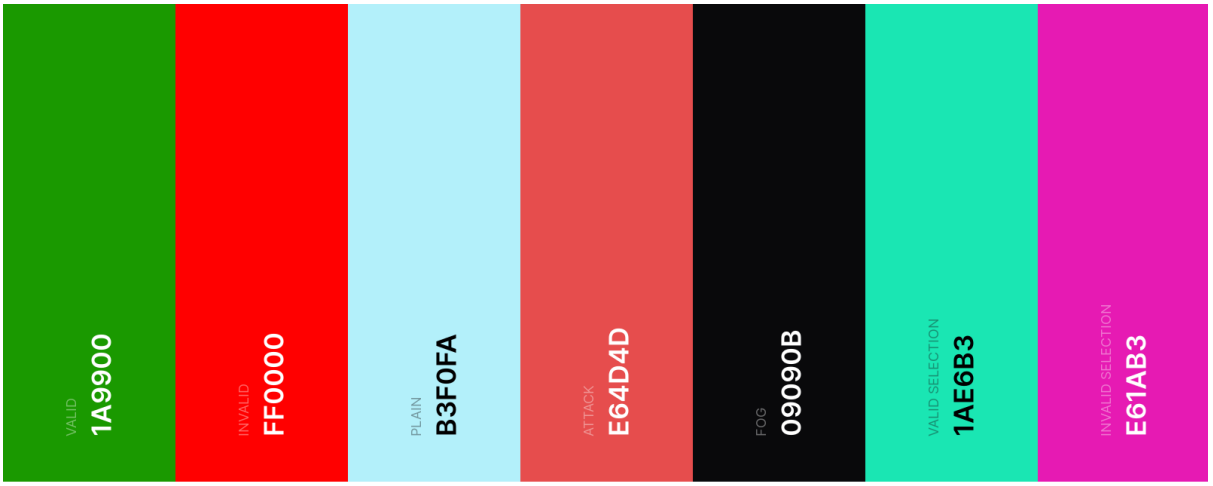
The main types of button style are either *interactable* or *not*. We re-used the above textures to indicate this, the one on the left is an interactable and the right is for information.

We also decided that any non-gameplay interaction e.g. settings will be displayed in the center of the screen. This is consistent throughout the game which makes the game more intuitive to play.

Most of the UI elements we based them off a resource pack called Golden UI, but the theme and layout of the game was based on Risk and Civilization.

4.3.1 Use of colour

We created this colour palette to indicate to the user what they can or cannot do depending on the highlight/colour of the tile they have selected.



4.3.2 Dynamic Cursor

The cursor was also designed in a way to give feedback to the user depending on what they can do in the current state of the game. Below they are in the order; Standard, In Click, On Hover.

5. Software Engineering

Throughout the development process, we successfully utilised a variety of software engineering processes and principles to ensure that our end product would meet our requirements.

5.1 Research Phase

At the start of the project, we reserved time to plan our game before we began any programming. During this time, we tried to figure out exactly what we wanted from the game and researched how to implement it. We created a requirements document that utilised MoSCoW prioritisation, providing us with strong foundations and a clear direction. We also did research into how the UI should be laid out, investigated how players should interact with our game, and researched game engine design- looking at common patterns and structures.

5.2 Project Management

As well as spending time on research before development, we also came to the conclusion that it would be best to manage our project through GitHub instead of GitLab. This allowed us to access features such as issues, actions, project boards, milestones and code reviews for pull requests. These features helped streamline the whole development process.

We set up four continuous integration actions at the start of development, each checking a distinct part of the code we produced. The ‘build’ action ensured our program would successfully compile. The ‘test’ action ran all of our JUnit tests, to make sure any modified code still passes. The ‘lint’ action checked for code style errors. The ‘doc-gen’ action guaranteed that Javadocs could be produced and warned of invalid Javadoc formatting. These actions would run on every pull-request, preventing merging until they all passed. This ensured that any code we wanted to merge into the main branch would meet our standards and would not break any pre-existing functionality.

We also set up an action to mirror our branches from GitHub to GitLab, ensuring that our code would be in the correct location for submission.

Furthermore, as we could not meet face-to-face, we utilised a Discord server to allow us to talk and share screens. We also created separate text channels for each component of the engine and game, allowing us to organise our discussions.

5.3 Agile Methodology

As a team, we decided to go with an agile approach to development.

One of the ways we did this was through scheduling two meetings a week, in a scrum-style stand-up format. In this, each member of the group took turns to explain what they had achieved since the last meeting, and what they would do by the next meeting. This creation of short-term goals helped us to break up concepts into manageable tasks, while the frequent reflection ensured the whole team understood the current state of the project.

We combined these twice-weekly meetings with weekly sprints, helping us to define middle-term goals. We utilised GitHub's kanban boards, creating issues for any features we wished to add in the sprint. We also had a larger project kanban board, and used milestones to set internal deadlines, to allow us to easily check our overall development progress. By creating issues for any required future functionality, as well as for any bugs we encountered, we could effectively manage our time and track development.

In some cases, when working on more complex problems, we also used paired programming (via screen sharing). While one person programmed, the other would review the code as it was typed- allowing us to quickly spot potential errors in the functionality, as well as any deviations from our standards. This helped us produce solutions to complex problems, while also increasing the quality of the code.

Furthermore, before a branch containing a new feature could be merged into the main code, we required the changes pass a code review stage. During this, the new code would be checked with the continuous integration checks, as described in 5.2 Project Management, to ensure that the new functionality would not break pre-existing code and to make sure it meets our coding standards. Once a pull request passed these checks, at least one other group member would read through and execute the code. During this review process, the reviewer would comment on the new code and suggest improvements or ask any related questions. Once the creator of the pull request responded to all of the comments, it would need to be reviewed again. This process- of running the continuous integration checks, reviewing the code, and responding to comments- would continue in an iterative manner, until the reviewer approved the code for merging. All 190 of our pull requests went through this thorough process- allowing us to ensure that we found problems before they made it into the main branch, where they could have potentially caused major and hard to find issues.

6. Risk Analysis

We adopted scrum techniques, working in weekly sprints - creating issues and working to solve them. An inherent risk with this strategy was that some team members could fall behind on their issues for the week. Furthermore, there was the risk that we could have too many issues open simultaneously, which could potentially become overwhelming and confusing. To mitigate this, we clearly prioritised issues, allowing each team member to manage their time to tackle the most significant problems and features. Our twice-weekly meetings also ensured we reviewed our current progress, and any team members who had fallen behind or encountered problems could easily access help and support from the rest of the team.

One potential risk with our development process was that different areas of our main code base could become fragmented and incompatible, as team members worked on different areas of the code. To help prevent this, we ensured we had a thorough review system in place, in addition to continuous integration checks, which ensured all new code correctly worked with pre-existing functionality. This review process also had the additional benefit of allowing other team members, who may not necessarily be working on the area being reviewed, to gain an insight into how different functionality was being implemented. This meant that even though a single team member may be specialising in a particular area, the other team members gain an understanding of the code- reducing the chances of future functionality being implemented that breaks the newly introduced code.

Another related problem was the potential risk for large merge conflicts, if multiple team members modified the same files. To help prevent this, we made sure to clearly define tasks and communicate where modifications would have an impact. This allowed us to mostly avoid overlapping edits, which would result in conflicts.

A large risk was the development of the game engine, as a significant amount of time needed to be spent developing it, and the team could become heavily reliant on the engine needing to be built before the game element could be started, as the game required several abstract classes and interfaces, from the engine, to function. To mitigate this, we had to carefully plan the order in which different aspects of the engine and game were created- to avoid having to rewrite code or delay development. We also decided that the potential benefits of creating our game engine outweighed the potential downfalls, as the engine provided us with a framework that allowed us to speed up the development of the game components, while also increasing our flexibility when needing to modify aspects of the game.

Feature creep, through continuously adding new requirements and functionality, was another risk we faced. We decided to mitigate this through the creation of a detailed set of requirements that could act as a solid foundation for the game. Additionally, our regular meetings allowed us to effectively communicate and debate any new additions, allowing all team members to understand any additions. Furthermore, our game engine provided a lot of flexibility when designing the game, so any additions did not need to completely rework pre-existing code.

One potential risk was the limited time frame, of 12 weeks, we had to fully develop the game and complete all documentation. To avoid this risk, we made sure we thoroughly planned the project and

had rough time scales for completion of certain aspects. We also made use of internal deadlines to ensure certain parts of the game and report were complete, allowing us to focus on other areas that needed attention.

A smaller risk was ensuring the game ran on a variety of different operating systems and hardware. Since each team member had slightly different setups and ran a variety of different operating systems, we could easily identify any weaknesses.

An obvious risk at the time of development was the ongoing COVID-19 pandemic. To mitigate against this, we ensured that all work and meetings were done virtually. We made sure to effectively use the different services provided by Discord, GitHub and Google Drive to ensure continuous collaboration and communication.

7. Teamwork

Throughout the project we worked well as a team to complete our goals. We used Discord extensively to plan and carry out meetings. We carried out formal stand-up meetings twice a week, in addition to frequent additional meetings in which we would all work simultaneously on our individual code- allowing for the free exchange of ideas and assistance. We structured the voice channels within Discord so there was one main channel, and several breakout channels. This allowed us to freely ask each other questions, and breakout and reconvene whenever necessary. This also helped encourage the use of paired programming, which was helpful when constructing more complex code and led to higher quality programming. Furthermore, individual members would often organise smaller meetings during the week to discuss speed bumps or problems in specific areas- helping increase access to assistance from other group members.

Another key aspect to our team work was our use of code reviews. These ensured that any implementations were understood by multiple members of the team, while also helping to encourage constructive criticism in an appropriate setting.

We also allowed individual team members to suggest which tasks they would like to undertake each week, when splitting up features to implement. This allowed some team members to focus on areas that fitted their personal strengths or interests, which helped further boost productivity. Additionally, we always tried to encourage the use of group work when larger tasks, which could not be further decomposed, were undertaken.

Our strong initial planning ensured that all team members understood the direction of the game, with debates normally centering on specific implementation details, rather than larger gameplay changes. These debates were always held in a constructive manner, and typically resolved quickly as each individual involved grew to understand the processes behind the other available options.

Altogether, we worked very well as a team. We ensured that we worked in an efficient manner that played to individuals' strengths, while also trying to ensure all members were involved in decision making and development throughout. The result of this teamwork is the strong final product we managed to produce.

8. Evaluation

8.1 Strengths

A major strength we had during this project was our ability to create clear plans as a team. At the start of the project, we spent time envisioning a clear idea which we could all work towards. We then effectively split this task up into more manageable parts, allowing individuals to focus on key areas. We also realised that some aspects of our desired game would take a longer time to produce, such as the renderer. Having spotted this from the start, it allowed some of the team to start coding right at the very beginning, while other members focused on defining the finer details of the game.

We also remained in constant communication with virtual team meetings. The frequency of these meetings allowed us to strongly work together, reviewing what we had done, planning what needed to be done, and allowed individuals to access help for specific problems. This was furthered by our use of tools like GitHub's issues and kanban boards, which allowed us to set clear goals as a team.

From the start of the project, our use of continuous integration and constant reviews ensured that we all had a strong understanding of each other's code- rather than being confined to the initial areas we assigned ourselves at the start of the project. This led to the team being extremely flexible, being able to read and utilise other's code with relative ease, due to the strong grasp we all had on the implementation of the project. Furthermore, the thorough review process we implemented encouraged the acceptance of constructive criticism, with code often going through our iterative review cycle several times before all parties are happy with the final result.

Another key strength we displayed was our adaptability and willingness to refine our project. One way this is shown was through our desire to apply newly learnt techniques to improve the code we produced. A notable example of this is with our adaptation to use lambdas within our code, which the majority of the team were unfamiliar with in Java. Through the adaption of these, initially unfamiliar, programming techniques, we were able to implement more advanced functionality while also increasing readability. Additionally, another way we showed our desire to improve our product was through each team member's willingness to support changes to code they created. One example of this was the implementation of our audio system. Originally, we used the JavaSounds library, but while planning our Week 7 demonstration, we found many flaws with the library- which our initial research had not identified. This led to other members of the group completely rebuilding and recreating the audio system using OpenAL. This was a major change which would not have been possible without the team's willingness to adapt.

8.2 Weaknesses

Our merge review process was a major advantage, but it did have some downfalls. This was because at certain times we would have too many pull requests open, and not all of them were reviewed in a timely manner- leading to some backlog. This was especially apparent towards the end of the game development process when we had many extra features being added. If we were to do this again, we would limit the number of allowed open pull requests which would result in older PR's being reviewed and not accumulating.

Even though our adaptability and willingness to add to our requirements was crucial to the quality and playability of the game, this led to problems towards the end of the game. When we had the basic game working, we began adding a lot of extra game features which massively increased the quality of our game. However, this also meant we had to ensure these additions did not interfere with pre-existing requirements. Once implemented, we also had to ensure all of our tests were up to date and functional. This took up a lot of additional time that we did not initially account for.

Even though we set up a GitHub action to check our Javadocs, we later realised that this check would pass if public methods and variables had no Javadocs on them. This meant that, at times, there were large swathes of methods and class level variables which had no documentation. This meant other members of the group using these methods did not always fully know what the method was doing. This was especially difficult when we were trying to debug the code to catch errors. To fix this, one of our members created a checkstyle to allow us to find the missing Javadocs and correct them. However, by the time we created the additional check, we had already unintentionally accumulated a large amount of documentation debt which had to be sorted. If we started again, we would have integrated this plugin in our continuous integration pipeline from the very start.

8.3 What Has Been Learnt

One thing we all have learnt is the importance of peer reviewing code. None of us had reviewed code at this scale before so it was a new experience for all of us. This allowed the team to have a consistent coding style throughout the development process. This meant the code was more readable and maintainable. We were also able to catch a lot of bugs and errors in our code because of this. Overall, this improved our games quality massively.

Maintaining a coding standard during the development was another important lesson from this project. This brought a lot of uniformity to our code as each of us had very different programming styles. We also followed the Javadocs standard when commenting on all of our code. Other small standards like using the same formatting tools were used. This allowed for the codebase to be more readable and integratable.

General time management was a skill we all learned. Following the scrum development technique of setting goals for each week and following through with them served us very well. Without following a strict timely pattern like this we would have severely fallen behind. Even though we followed the scrum pattern we still had to do more work in the last few weeks closer to the deadline. However, we did manage to finish in time with a high quality game. A lot of this is attributed to our time management.

8.4 Improvements

The underlying game engine could have efficiency improvements. First of all, our current design is largely single-threaded. An improvement could be added for thread-safe update contexts. In addition, our component lists do not use the most efficient design. At the current game map size of 51 it is not a problem, but increasing it to over 250 puts the game to its knees as 162,000 tiles need to be monitored. With our current design, whenever an object spawns, gets destroyed, gets enabled, or disabled, the whole list is invalidated, requiring a full reiteration over all objects in a given scene. Improvements to the current design would be use of sets and maps. This way, change of a single object's state would not require a lot of computation.

If we were to make a fully-fledged engine, we would improve our tooling. The Blender integration could have been made more robust, with Java based class parser. Currently we use a set of rough heuristics in a Python script, trying to collect a list of components in game. Having a dedicated UI builder would make the experience of building our user interfaces quicker and more intuitive, because currently, every tiny change requires restarting the game.

In terms of graphics, a post-processing effect stack could have been implemented. This would allow for additional opportunities to open up, such as bloom, depth of field, exposure correction, screen-space reflections, and other visual effects. It would also lay the groundwork needed for shadow mapping, which would also add extra depth to the game.

In networking, the message system could be converted to use User Datagram Protocol (UDP), potentially allowing for lower network latencies. Also, selective per-player object synchronisation could be implemented, making wall hacks physically impossible. This potential addition is described in detail in 3. Software Design.

Another improvement would be to create a smarter AI for the game to make it more enjoyable. Our current AI is relatively basic- it uses probabilities to choose what move to do. This means that sometimes it might not take the 'best' course of action, like expanding rather than attempting to defend its Capital. Furthermore, if there are only AI players left there is a chance the game could lead to a stalemate- or at least take a very long time to complete. To improve the AI, one method would be to use more advanced techniques, such as utilising something similar to a Monte Carlo Search Tree, to allow it to pick actions which gives it a better chance of winning- thus making it play more like a human. This in turn would make the game more enjoyable for users as they would have a more responsive AI player to play against.

A further improvement would be to have more choice in hosting a game. Currently the game is set to a map size of 51 with up to 6 players (either human or AI). An improvement would be that the host would be able to change the size of the map, and to allow for a specific number of human and AI players to be specified. This would improve replayability due to the additional customisation. In addition, allowing the host to define the seed used to generate the map would provide greater control to the host, and encourage players to try and play the same map multiple times.

In terms of tiny details, the game could have been made more lively with more sounds added to various objects of the world. Currently, we have a few spatial sound effects in the blacksmith building, and this could be extended to all parts of the game.

If we had more time available, we could increase the diversity of gameplay by introducing concepts like factions, that introduce different themes for your models and gameplay modifiers. This would increase the diversity of the gameplay and provide more unique experiences to the player. This would also encourage players to change their playstyle based on their own abilities, and the abilities of their opponents.

9. Summary

Overall, through careful planning, clear organisation and a strong understanding of what we wanted the final product to be, we worked well as a team to produce an impressive game- all within a tight time frame.

We are extremely proud of our ambitious game, as it satisfies all of our key requirements- it is a fully networked strategy game that provides a complete user experience, while being genuinely fun to play.

10. Individual Reflections

11.1 Craig Wilbourne

The first major tasks I undertook was the formalisation of requirements. This meant that I had a strong understanding of what we wanted from the game and could help ensure team members were aware of the different parts of the game we needed to implement.

I then began the planning and implementation of the input system, within the game engine. Despite this being an unfamiliar area for me, I feel I successfully managed to create a robust, yet adaptable, system that effectively delivers a convenient interface for accessing and managing input. I am pleased that the core of my input system remained virtually unchanged throughout later development- with the only additions being ones that were planned in advance- showing how I tried to carefully consider all the future functionality we would require.

After completing the input system in the engine, I mostly moved on to developing the actual game logic. One area I focused on was the implementation of buildings, and the wide variety of functionality they required, and tile map logic. The building and hexagon tile logic I developed also linked closely with the Player, HumanPlayer and AiPlayer classes, which I also helped create in conjunction with other members of the group. When developing classes like Player, we successfully utilised paired programming to help ensure code correctly functioned and interacted with other components of the system, while maintaining high standards.

I also assisted with the implementation of the UI within the game. As well as implementing functionality, such as the pause menus, I also spent time overhauling the UI: replacing the confusing interfaces we produced during initial development so they would closer match our planned interface designs. I tried to ensure these improved overlays, such as the building shop and the upgrade shop, were more intuitive to use, provided clear feedback to the user, and conformed to our HCI principles.

Due to my strong understanding of the requirements, I also ensured that key features that had been overlooked, such as attack cooldown, costs for attacking and selling, and placing different types of buildings, were fully implemented, and provided feedback to both the human players, visually, and AI

players, via their logic. Furthermore, I also frequently undertook the task of refactoring outdated code to improve readability while also upgrading and simplifying logic.

I also tried to frequently review the code of others. One example of this is with the code for the AI players' A* pathfinding, in which I ended up finding and commenting on over 150 issues. By the time the code was merged, thanks to the review process, it was at a significantly higher standard.

By the end of the project, I mostly focused on report writing, as well as implementing various code changes and general bug fixes.

Throughout the project, I felt we worked well as a team, with all members being happy with the direction of the project. Tasks were distributed to account for coding ability, previous experience, and interests - helping us to maximise what we could achieve in a limited time frame. The only flaw with the team was that some members did allocate a lot of their time to the team project, against the advice of the group, which could be seen as unrealistic for others to match. This did help ensure we had an impressive game, but also set extremely high expectations.

Overall, I am pleased with my contributions towards the team project. I feel that I have tried my hardest to organise the development of the project, contribute to the code, and produce a final product we are all proud of.

11.2 Oscar Lindenbaum

My key contribution to the project was the basis of the networking, improving the game experience in areas like gameplay and UI in addition to creating the serverless API for clients to find any WAN games available. Initially I created a basic client-server model and wrote tests for the main usages, then I adapted the system towards the game. The idea was to mark any values that needed synchronisation within its class and then the authoritarian-network update its clients only when needs be, with only the values that had changed to minimise data transmission and to optimise speed. I first created this using 'field-separators' but this was unneeded when we transitioned to bit masking the message and also making 'Syncables' implement their own optimal type of serialization using streams. This allowed us to have a big speed increase and the system for registering syncables was as simple as can be: Create a field that extends *ISyncVar*, and place it inside a *NetworkableComponent*. Later in development we added the feature to register and trigger events in a bi-directional manner between the client and the server.

At a later point in the project, we encountered errors with some platforms not being able to communicate with the serverless API due to what I believe was certificate rejection, or TLS issues. To solve this we moved it from a serverless instance to be hosted on a Droplet on DigitalOcean. During the transfer I also implemented additional API endpoints for joining a game by 'CODE' and for updating and retrieving game settings remotely. In addition to this I created a way of storing and retrieving local user settings, which can be modified and persisted across game reloads.

I also worked on creating assets and making the game UI, this also meant making the UI intuitive to use and responsive to the current state of the game. It was also important to make it as homogeneous across all states. This was harder than expected as our game is in a 3D world, meaning light and depth was important but more the case that it was hard to position Objects correctly in the 3D space.

A concept that took a while for me to understand was the Component system. The component system is where a parent *GameObject* had many components which could see and communicate with each other, i.e. a shared *Transform* Component and *Renderable.* I got confused when the hierarchy became more complex with *GameObjects* having both *Components* and other *GameObjects* as children also. As the project advanced I got to grips with the system quite confidently and understood the difference between *Frame/Fixed* updates and the execution at different times.

Further into the project I realised our JavaDocs were not consistent and our naming didn't always follow our styling. To easily help everyone find and fix their mistakes, I adapted the *google-checkstyle* document to exactly fit our specification. This meant you could check all files for the correct naming conventions and documentation in one command.

What I think our team did well was the continuous integration and communication throughout the entire project. As we utilised GitHub extensively, no single piece of code was not agreed upon solely by the creator meaning everyone knew what features were introduced, how it worked and was soon to be merged to the main game. The team also helped each other across the development when anyone had a problem or were not sure how to solve a bug, this included spending many hours on a discord call sharing screen for most of the day until a state we were happy with was reached. We were late to start actually coding the product as we decided it would be better to plan in detail and write much of the supporting documentation before we began as this would also mean better code continuity and make it easier to piece together. There were also no major conflicts or disagreements within the team and we were all passionate about creating a great final product which made it easy to stay focused.

Compared to the start of the project I would say I was a fairly confident Java programmer, but now I would say I am much better equipped and also think of the bigger impact by writing good code from the beginning rather than having to change it all later. I also found out about different concepts in programming such as the *singleton pattern* and *enum constructors* which I did not know about before and will use in the future. I would also say the same for my git capabilities, now I can confidently use github, do code reviews, pair programs, and also use the CLI to a greater extent and power by the use of *cherry-picking* or *stashing* and maintaining a better repository.

Overall I am quite proud of the result we have achieved and also in the way we have achieved it, although working at a screen for long periods is especially hard for me due to personal reasons, I do not think we could have done a better job in person.

11.3 Aurimas Blažulionis

My main code contributions to this project were the 3D Vulkan renderer, file resource manager, co-designing networking code, optimising it, building higher level network abstractions, building a UI framework, Blender and glTF integration, adding random elements to map generation, visual effects, testing framework with network tests. In addition, I set up a continuous integration system we used throughout the project, created 3D models visible throughout the game, and were hands on discussing with the team, helping each other out.

I learnt a lot from making the renderer. It was difficult at first, with many key concepts needing to be learned. And it was very slow at first. But, after the first triangle was drawn, progress started to be faster and faster. I have always wanted to make a 3D renderer, but was always fend off by the amount

of moving parts needing to get just right. Taking this project as an opportunity to learn low level 3D graphics was extremely challenging and rewarding. Creating a physically based renderer is quite a feat, and I look forward to delving deep into more complex areas of computer graphics.

Having prior networking knowledge, I helped Oscar with building our multiplayer solution. It was a tedious process trying to get it just right. I have picked up quite a bit of skill in code reviews, applying various optimisation techniques and overall looking at the very big picture of a multiplayer game.

Building the Blender and glTF integration, I learned an incredibly useful skill of writing Blender plugins. In addition, I learned quite a bit about glTF, and the way such file formats are laid out. I liked the idea of how everything was flat, yet still able to have a hierarchical tree structure constructed out of it.

In terms of UI, I very much underestimated it. UI transformations and hierarchies are hard, especially when one wants to scale objects relative to their parent bounds, maintain aspect ratio, align to different sides of the parent, and more. For a good while, we were using a semi-broken implementation of TransformUI, which then took me a whole day and night trying to make the implementation behave sensibly.

This was my first programming project with a team of this size, and it was quite an experience. As a team, we seem to have split up the work fairly, relative to our individual skill ceilings. Given game development is my hobby, I feel like I have been pushing the team to be more and more ambitious each day. They probably hate me for this, but I feel like raising the standards as a whole is a more constructive and sustainable way forward than lowering your own to match others. Of course, I understood that not everyone was as passionate about game development as I was, so I had no expectation of the team to devote the same amount of time to the project. I also very much understood that everyone had different skills and their ceilings, so what mattered to me was overall effort. In the end, I was very lucky to get the team I did, because nobody was lazy, nobody was avoiding taking up work, nobody caused any conflicts. We all worked well as a team. I have never done code reviews at this scale. They took time, but were definitely a wonderful idea driving code quality up.

11.4 Nathaniel Lowis

Overall, this team project has been a major challenge in time management, working in a software development team and implementing concepts which I have learnt in lectures to a real life problem.

I initially worked in collaboration with others on the early structure of our Hexagon Map, specifically planning and implementing useful features for the game. I then started to write test outcomes for our test plan and a basic risk analysis. Although my initial role was not primarily involved in creating the engine, I did create the original audio system for the game. Once the audio system was in place I then created the original Player, HumanPlayer and AI classes for the game. Whilst putting this in place I was working from other people's developing code and learning how to use our networking and UI components of the game. I focused on these three elements until the Week 7 demonstration. Unfortunately my audio system was scrapped after week 7 because JavaSounds was found to be incredibly buggy. Instead, the audio system was rewritten using OpenAL. My focus therefore changed as I started to rewrite and rework the AI, giving them a full range of actions which would be available to the user. I was able to create how the Capitals would initially be placed and furthermore I created new AI classes to be used by the computer. Throughout the project I have been helping in debugging and using the Player class. One of my largest contributions has been to check whether an attack was valid on the server and implementing what the state of Player and Building would be after a successful attack. As we neared the completion of our project, I helped write the first draft of the software engineering and teamwork sections of the report.

In this project, I have had to deal extensively with how to create AI which is enjoyable to play against, works smoothly and gives an acceptable amount of challenge to the user. This has extended what I have learnt in AI lectures (which focus more on optimality problems) whilst still using the ideas I have been taught (like the A* algorithm) in a practical setting.

Our team worked well together. Everyone put in a similar amount of effort and everyone worked to the best of their ability. We had very few conflicts partially because we were good at keeping one another informed of changes which were happening. The only issues which did occur was due to the pandemic as people were working at different times, which consequently made it difficult to arrange one to one calls which were sometimes needed. A major way the team worked well together was the use of pull request reviews. This improved the standard of my code before it was merged into the main branch by both creating a more readable and more efficient code. This meant I have become more conscious of how my code reads to other people and has meant I have learnt about new Java structures which I did not know about before.

My foundational understanding of Java and Git has been significantly improved during this project. I have learnt a lot more about the language and the different uses of it. I have learnt to use Java Streams, enums, lambda functions and private classes and private interfaces within a public class. I have learnt how to use Maven to create and run projects. Finally, I have learnt more Git features (like git merge and cherry-picking) and the different features that GitHub has like Pull Request reviews and automated actions.

To conclude, I have thoroughly enjoyed this team project and believe it has prepared me for being able to work in teams in my future workplace environment. I have become more confident using Java and Git which will be helpful for future projects. I have always been able to work well in a team but

have honed these skills in the unusual situation due to COVID-19. I am incredibly proud of the fully workable and enjoyable game that we have created and my part in this process.

11.5 Harry Stoltz

My main contributions to the project were the core of the engine (GameObjects, Components, Scenes and Transforms), the audio system and also managing the lobby system of the game. For the documentation I did the software engineering section and also did a section in software design covering the audio system.

The engine design was a component and object design that was influenced by the Unity Engine which I have a bit of experience using. It was designed so that any new elements, be it audio, renderer, input, etc were as easy to integrate as possible, which was done by simply adding any calls that needed to be done every frame to the main loop. Components were designed to implement any number of the update interfaces which would be called at the various stages of the main loop depending on what scene the components were in. I wrote the initial implementations of Transform and TransformHex making use of JOML for all of the maths that was required, which have all been built on since.

The audio system was originally coded by Nat, however it was done using the JavaSound library which has a few bugs in Java 8, including an issue where a thread was left hanging at the end of execution. This issue prompted us to switch to a different library, and we decided upon OpenAL which had bindings in lwjgl which we were already using. I chose to do this work and designed the audio system which manages a pool of OpenAL audio sources that are dynamically attached to the AudioSource components in the current scene depending on whether they were actively playing sound and the distance of the AudioSource from the AudioListener.

The final major contribution from me was the lobby system. The first thing that I did for this was implementing a wrapper for the serverless API that Oscar wrote so that we could add, delete and get all of the hosts that were active. This was all done asynchronously making use of callbacks to handle the results of the requests. After this I wrote a UPnP utility class that could be used to automatically port forward for us so that we would be able to create lobbies and play over WAN. I also wrote a host discovery class that used UDP broadcasting so that players on a local network would be able to connect to each other without having to share IPs. The Lobby component then tied all of these separate parts together and also generated the UI that would be shown.

As a team I think that we worked very well together, everyone was trying their hardest and did as much towards the project as they could. We never had any serious conflicts or arguments and generally agreed on most design choices and implementation details. As a group we always attended every meeting (except for the times when the designated time conflicted with some other serious commitment) and would usually stay on call after the “meeting” section ended to work on the project and bounce ideas and problems off of each other.

Over the course of the project I have learned a lot. This was the first time that I have worked on a codebase this big with a team of this size. I’ve come to grips with a lot of Git features and have become comfortable with using it for a big project. It was also the first time I have used something like Maven for managing a Java project and I will definitely be using it more in the future because of how easy it makes adding libraries and managing code.

11.6 Leela Muppala

My main contributions to this project were the game logic section. This included making the hexagon map, the hexagon tiles, features of the buildings and UI elements.

The making of the hexagon map took the most effort from my side. This was partly because it was the first object that was made in the game. This was challenging because of all the different approaches to making hex grids and their correlation to coordinates. Since this was towards the early stages of the game development, most of the team didn't have a clear idea of how the map would work. I came up with a simple way of spawning a map of variable length with an axial coordinate system that removed redundancy. Algorithms that calculated movement around the grid i.e.adding, subtracting the coordinates became much cleaner because of this system. The axial coordinate system, however, complicated the map storage. To simplify this, I made an algorithm which according to the size generated a 2D array of hexagonal tiles, filling the unused spaces with nulls. This whole class was encapsulated in getters and setters so that the rest of the game didn't need to know the map storage logic.

Continuing my work from the map, I went on to work on the buildings and their features. I worked with Craig to initially work out a framework and logic for the buildings. He continued to implement most of this framework while I worked on the smaller features. Implementing the attack feature in this area was quite prominent. I had to work out an algorithm that took into account the player stats when determining the success/failure of an attack. We wanted to add a non-deterministic element to the attack feature to ensure a fair game. A simple dice rolling algorithm that took into account the players stats but also added a random element worked well, having tested it several times. I also worked on smaller features such as selling buildings and destroying them.

Towards the end of the game, I worked on the User Interface to make it more intuitive to the player. This was quite different from all my previous work but I was able to transition into it easily with Auri's help as he worked on this significantly. I worked on elements such as highlighting the valid tiles where the user can place buildings, highlighting the tiles with buildings that are attackable and highlighting all the invalid tiles.

This was a first for me as I had never worked on a project with a team this big. This was a very positive experience for me as I learned a lot from my team members through code reviews and working with them. I can use Git and all of its tools with a lot more ease now. Collaborating and implementing code from scratch is also a skill I've picked up.

The team worked very well together. We were all passionate about making a good quality game. We shared the work fairly according to each of our skills and tried our best. We would work together every week by staying on discord calls and helping each other out. Overall the team was very supportive and helpful.

11. Quantified Evaluation of the Contributions

Team Member	**Percentage contributed**
Aurimas Blažulionis	16.6%
Oscar Lindenbaum	16.6%
Nathaniel Lowis	16.6%
Leela Muppala	16.6%
Harry Stoltz	16.6%
Craig Wilbourne	16.6%

The above values have all been agreed upon by each team member.

12. Appendix

12.1 Coding Standards

We closely followed the AOSP coding standards throughout the project (https://source.android.com/setup/contribute/code-style).

For example, some of the key features of AOSP:

- Variable naming convention.
    - Depending on the variable, different prefixes are used. For example private variables are denoted by an ‘m’, and public static variables start with an ‘s’.
- Indentation.
    - Rather than using tabs, 4 spaces are used per level of indentation.
- Variable scope.
    - Variables should be in the smallest possible scope.
- Line length limit.
    - Unless in specific scenarios, lines should be no more than 100 characters long.
- If statements.
    - If an if statement cannot comfortably fit on one line, it should have curly brackets.
- Log appropriately.

In order to ensure we met these standards, we used a GitHub action to perform a continuous integration check to catch any larger violations. Our rigorous peer-reviews also ensured that our standards were met. Furthermore, we created a stylecheck to scan our code and highlight any areas where we accidentally went against standards. This helped us to eliminate any deviations from standards, no matter how small they are.

12.2 Bibliography

Websites which we used for inspiration or utilised within the implementation:

https://source.android.com/setup/contribute/code-style - AOSP Standards

https://github.com/low101043/aiProjectComputer - Code modified to be used for the A* Player
https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22 - Used for Inspiration for AI Players
https://www.redblobgames.com/grids/hexagons/ - Used to inform us of how to create and use a hexagon Map
https://www.gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/ - Used for testing the A* Algorithm
https://stackoverflow.com/questions/4816322/mapping-x-y-to-single-number-value - Used as a simple hashing program for 2 coordinates.
http://www.smokingdrum.com - FatPixel Font
http://www.myersdaily.org/joseph/javascript/md5.js - MD5 Algorithm Used in Serverless API
https://www.diffblue.com/ - Some Unit Tests were generated using DiffBlue.

https://vulkan-tutorial.com/ - Vulkan Tutorial used for initial renderer implementation.
https://github.com/SaschaWillems/Vulkan - Vulkan samples by Sascha Willems used when abstracting the renderer.
https://blackpawn.com/texts/lightmaps/default.html - Lightmap packing used for font atlas generation.
https://stackoverflow.com/questions/7692988/opengl-math-projecting-screen-space-to-world-space-coords - Screen to world used when converting mouse cursor to world space coordinates.
https://stackoverflow.com/a/26493311/13240247 - #include regular expressions used when preprocessing shaders.
https://www.youtube.com/watch?v=j-A0mwsJRmk - SIGGRAPH University - Introduction to "Physically Based Shading in Theory and Practice" used when researching physically based rendering.
https://www.youtube.com/watch?v=5p0e7YNONr8 - OpenGL - PBR used when researching physically based rendering.
https://github.com/Nadrin/PBR - PBR samples used when researching physically based rendering.
http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf - PBR by Epic Games used when researching physically based rendering.
http://www.thetenthplanet.de/archives/1180 - Normal Mapping without predefined tangents used when implementing normal mapping.
https://www.khronos.org/gltf/ - glTF format used when implementing the asset importer.
https://www.glfw.org/docs/latest/input_guide.html - used as basic guide o input with GLFW

https://freesound.org/people/BeezleFM/sounds/512133/ - Building Sell
https://freesound.org/people/InspectorJ/sounds/400991/ - Building Upgrade
https://freesound.org/people/kirmm/sounds/392180/ - Game Start
https://freesound.org/people/Tuudurt/sounds/258142/ - Game Victory
https://freesound.org/people/Akrythael/sounds/334916/ - Game Lose
https://freesound.org/people/adcbicycle/sounds/13900/ - Attack Failed
https://freesound.org/people/Isaac200000/sounds/184650/ - Attack Launched

https://www.free-stock-music.com/alexander-nakarada-battle-of-the-creek.html - Background Battle Of The Creek by Alexander Nakarada | https://www.serpentsoundstudios.com
Music promoted by https://www.free-stock-music.com
Attribution 4.0 International (CC BY 4.0)
https://creativecommons.org/licenses/by/4.0/

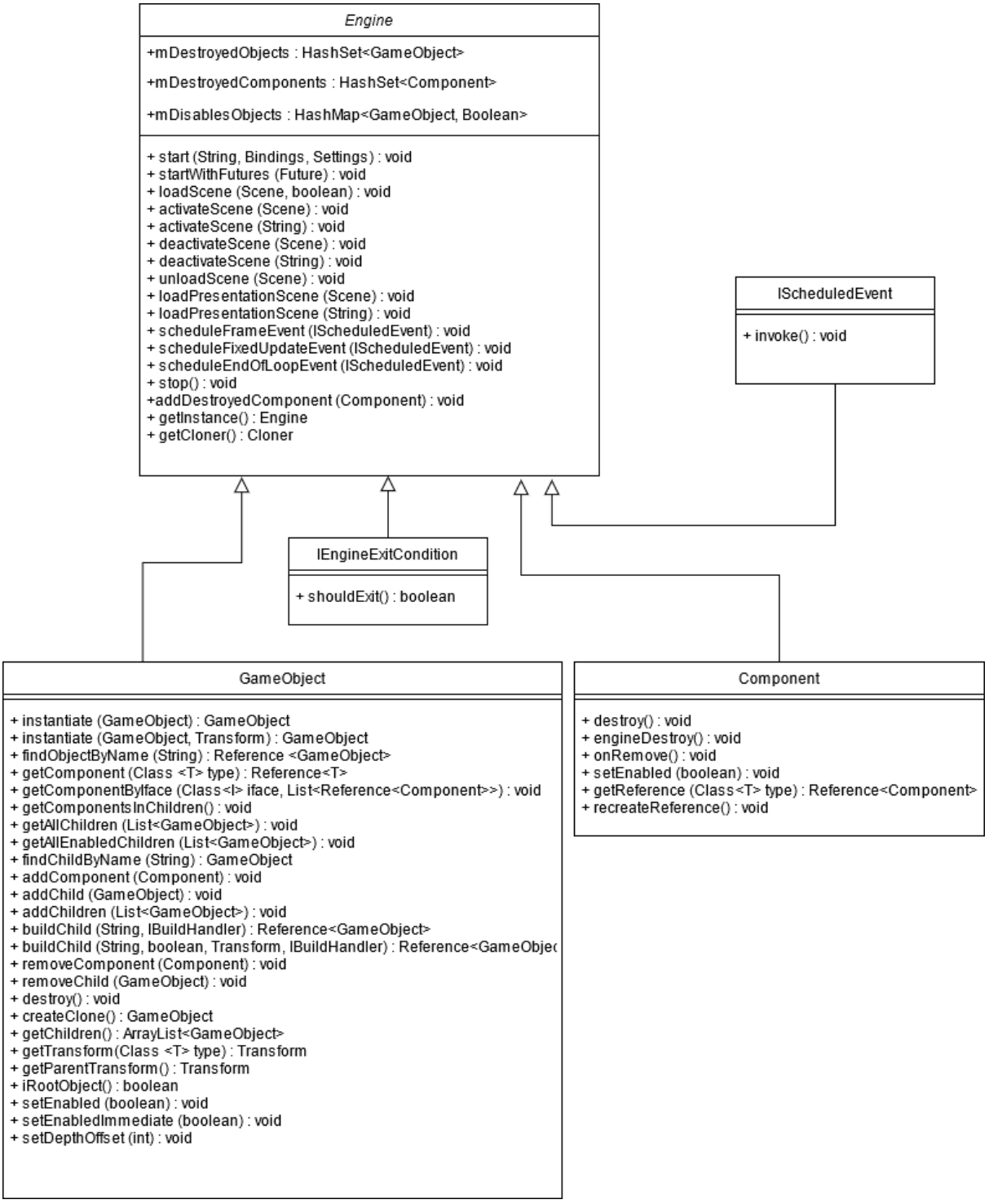
The following sounds were obtained rightfully from www.artlist.io
Attack Success - https://artlist.io/sfx/track/52658/destruction---wood-impact,-debris
Blacksmith Hitting Anvil - https://artlist.io/sfx/track/21499/blacksmith-hitting-anvil
Defence Failed - https://artlist.io/sfx/track/52688/dungeons---metal-gate-closing
Medieval Scraper Axe - https://artlist.io/sfx/track/34903/medieval-feel---scraper-axe-,-sharpening-
Medieval Sharpening Metal - https://artlist.io/sfx/track/34904/medieval-feel---axe-sharpening,-metal-shine-

12.3 Additional Diagrams

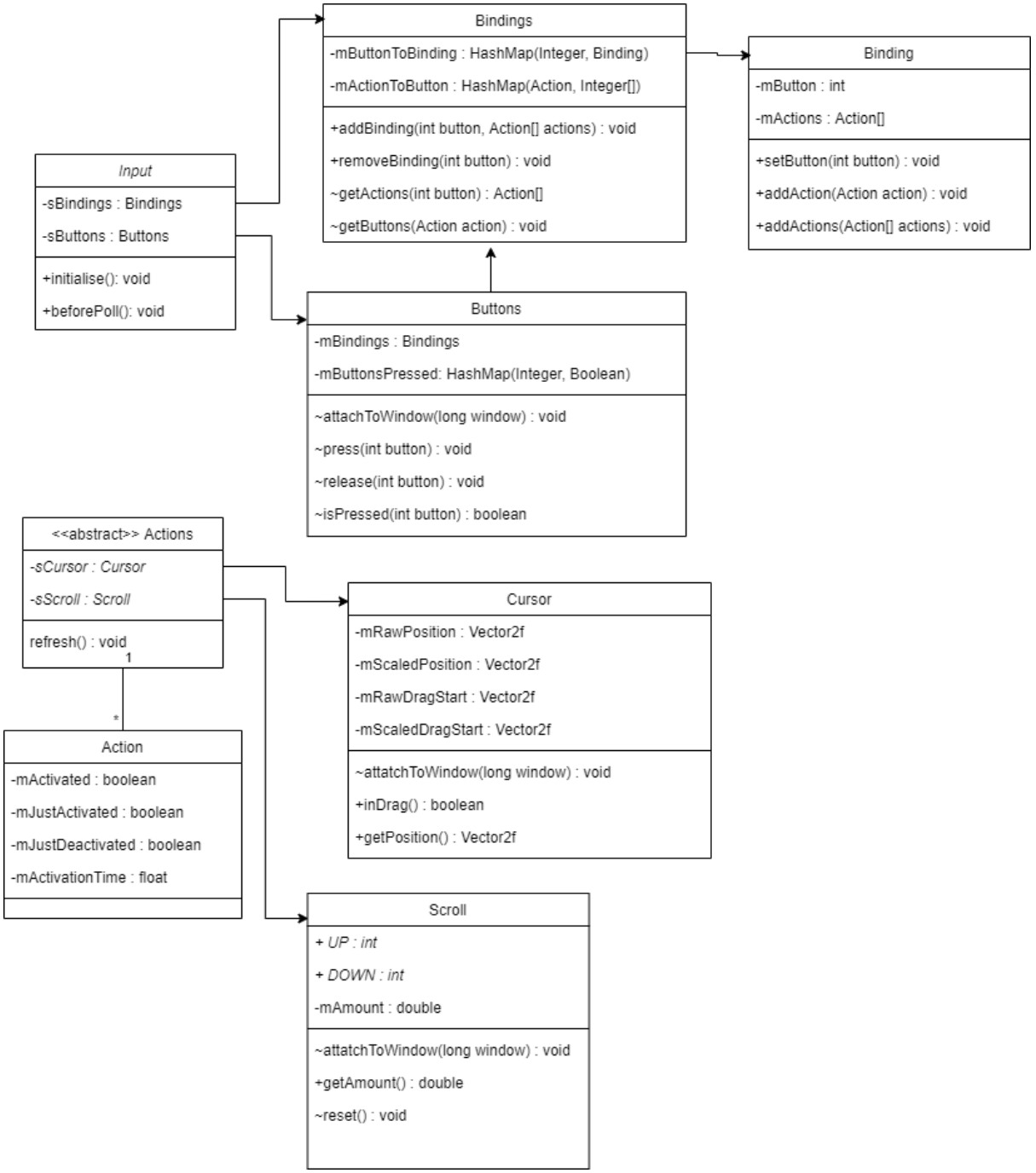
12.3.1 UML Diagrams

To aid readability, the majority of getters and setters have been removed from diagrams.

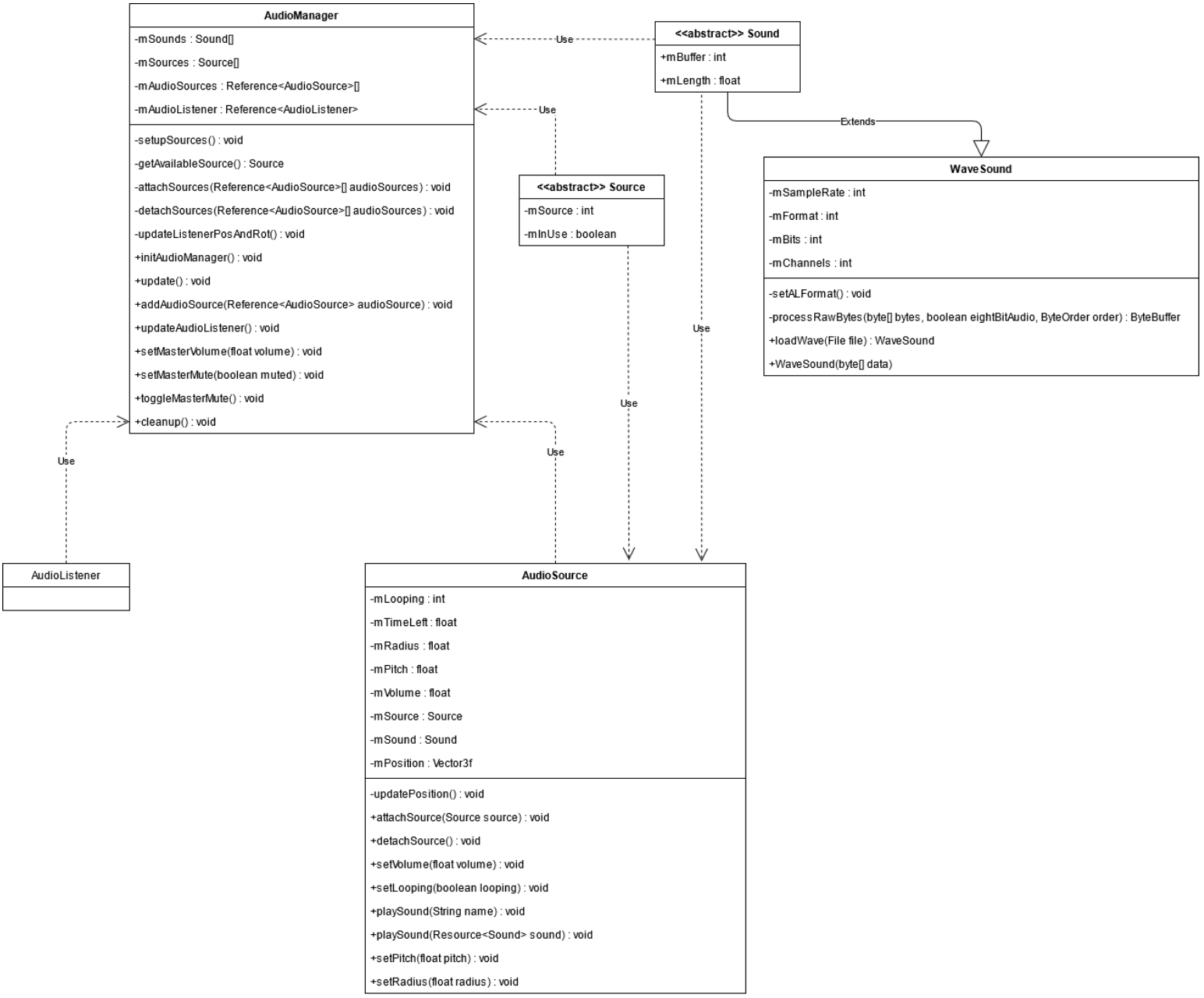
Overall engine system:



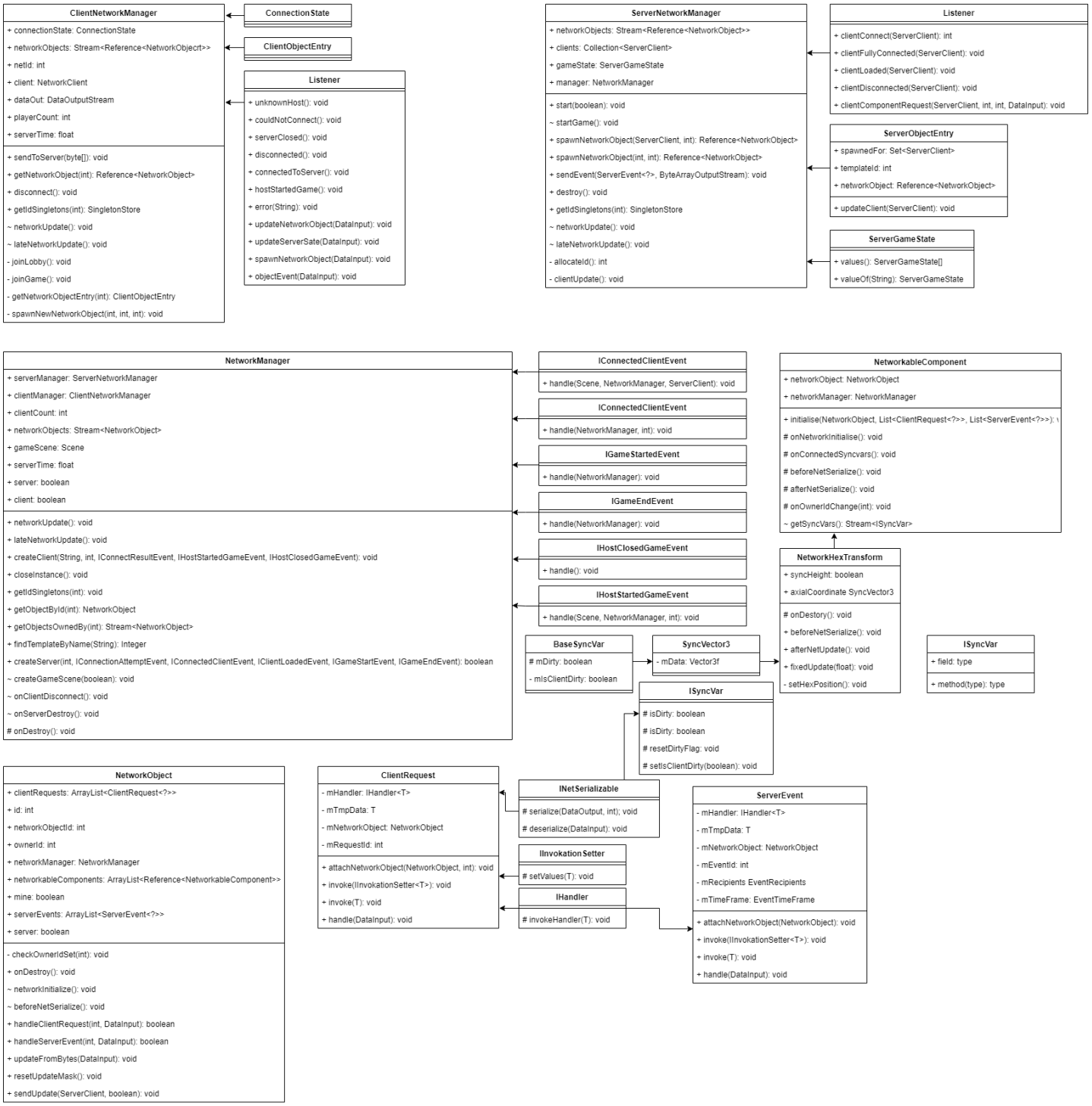
Input system:



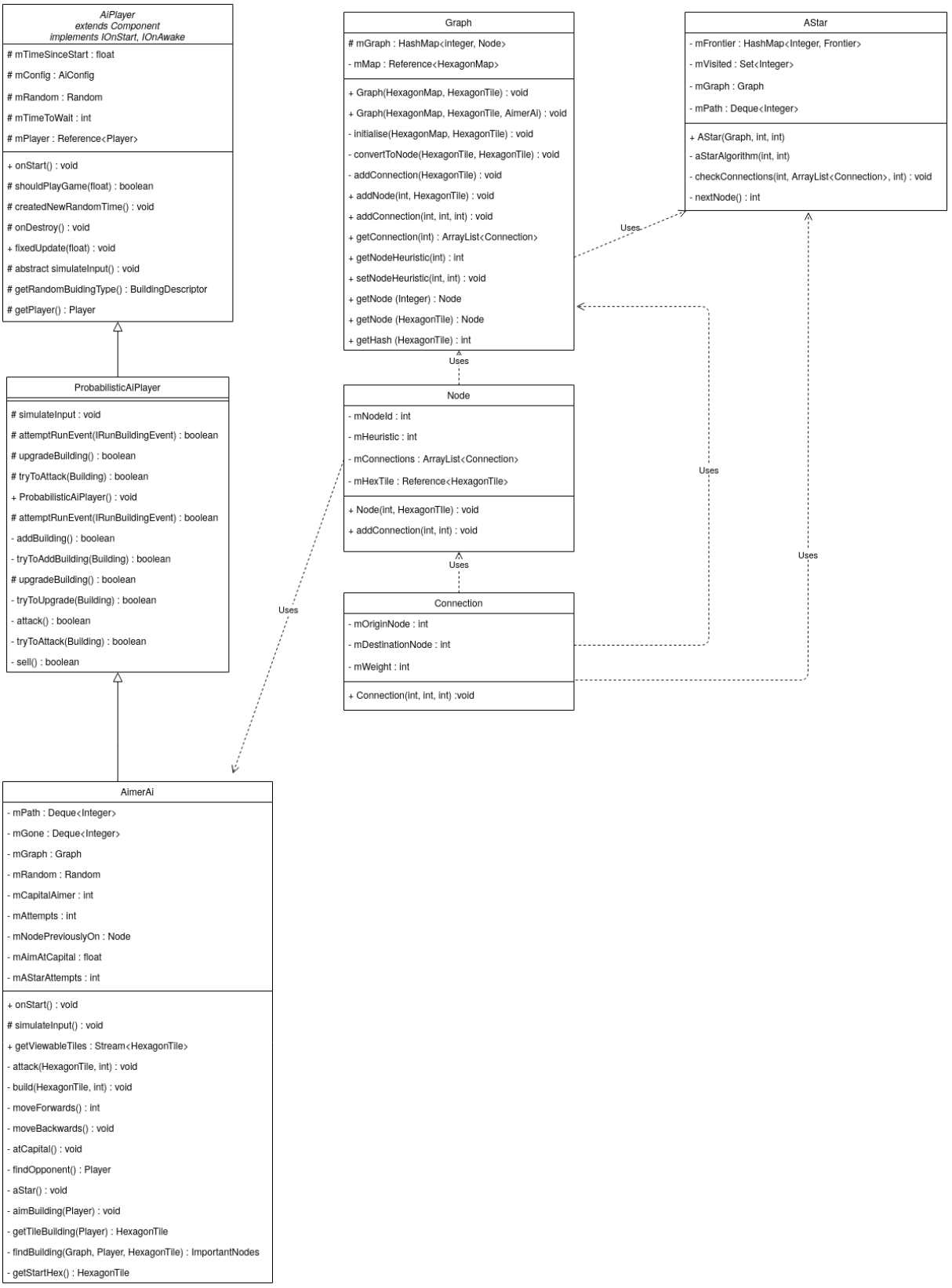
Audio system:



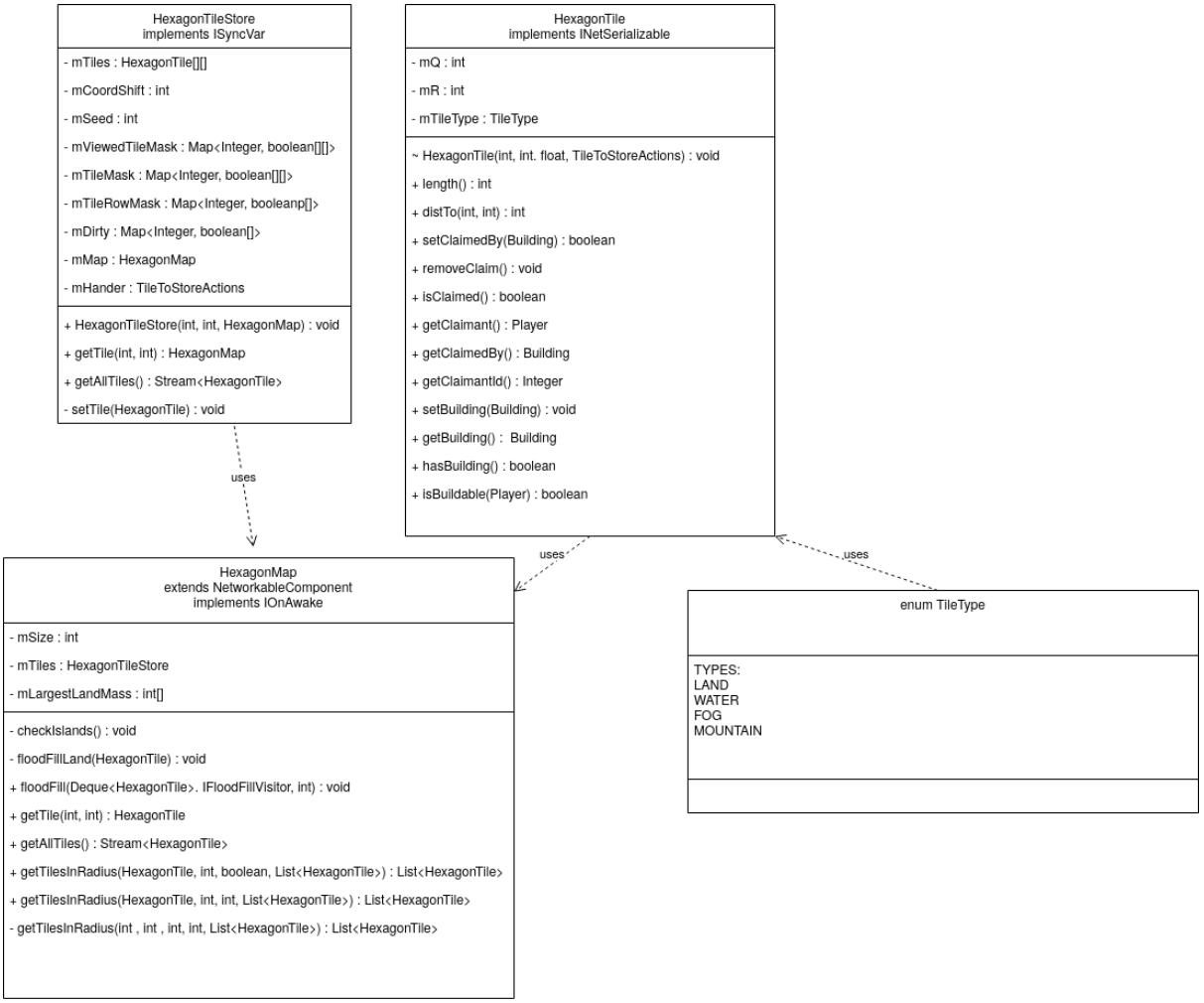
Networking system:



AI system:

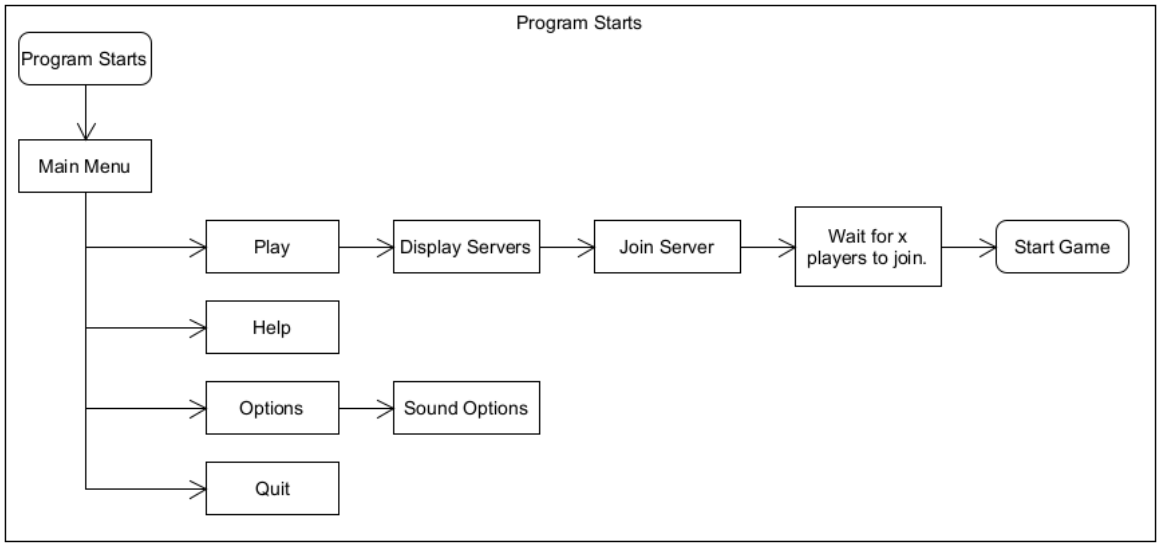

AiPlayer
extends Component
implements IOnStart, IOnAwake
mTimeSinceStart : float
mConfig : AiConfig
mRandom : Random
mTimeToWait : int
mPlayer : Reference<Player>
+ onStart() : void
# shouldPlayGame(float) : boolean
# createdNewRandomTime() : void
# onDestroy() : void
+ fixedUpdate(float) : void
# abstract simulateInput() : void
# getRandomBuidingType() : BuildingDescriptor
# getPlayer() : Player
ProbabilisticAiPlayer
simulateInput : void
attemptRunEvent(IRunBuildingEvent) : boolean
upgradeBuilding() : boolean
tryToAttack(Building) : boolean
+ ProbabilisticAiPlayer() : void
# attemptRunEvent(IRunBuildingEvent) : boolean
- addBuilding() : boolean
- tryToAddBuilding(Building) : boolean
upgradeBuilding() : boolean
- tryToUpgrade(Building) : boolean
- attack() : boolean
- tryToAttack(Building) : boolean
- sell() : boolean
AimerAi
- mPath : Deque<Integer>
- mGone : Deque<Integer>
- mGraph : Graph
- mRandom : Random
- mCapitalAimer : int
- mAttempts : int
- mNodePreviouslyOn : Node
- mAimAtCapital : float
- mAStarAttempts : int
+ onStart() : void
# simulateInput() : void
+ getViewableTiles : Stream<HexagonTile>
- attack(HexagonTile, int) : void
- build(HexagonTile, int) : void
- moveForwards() : int
- moveBackwards() : void
- atCapital() : void
- findOpponent() : Player
- aStar() : void
- aimBuilding(Player) : void
- getTileBuilding(Player) : HexagonTile
- findBuilding(Graph, Player, HexagonTile) : ImportantNodes
- getStartHex() : HexagonTile
Graph
mGraph : HashMap<integer, Node>
- mMap : Reference<HexagonMap>
+ Graph(HexagonMap, HexagonTile) : void
+ Graph(HexagonMap, HexagonTile, AimerAi) : void
- initialise(HexagonMap, HexagonTile) : void
- convertToNode(HexagonTile, HexagonTile) : void
- addConnection(HexagonTile) : void
+ addNode(int, HexagonTile) : void
+ addConnection(int, int, int) : void
+ getConnection(int) : ArrayList<Connection>
+ getNodeHeuristic(int) : int
+ setNodeHeuristic(int, int) : void
+ getNode (Integer) : Node
+ getNode (HexagonTile) : Node
+ getHash (HexagonTile) : int
Node
- mNodeId : int
- mHeuristic : int
- mConnections : ArrayList<Connection>
- mHexTile : Reference<HexagonTile>
+ Node(int, HexagonTile) : void
+ addConnection(int, int) : void
Connection
- mOriginNode : int
- mDestinationNode : int
- mWeight : int
+ Connection(int, int, int) :void
AStar
- mFrontier : HashMap<Integer, Frontier>
- mVisited : Set<Integer>
- mGraph : Graph
- mPath : Deque<Integer>
+ AStar(Graph, int, int)
- aStarAlgorithm(int, int)
- checkConnections(int, ArrayList<Connection>, int) : void
- nextNode() : int
Uses
Uses
Uses
Uses
Uses
Uses

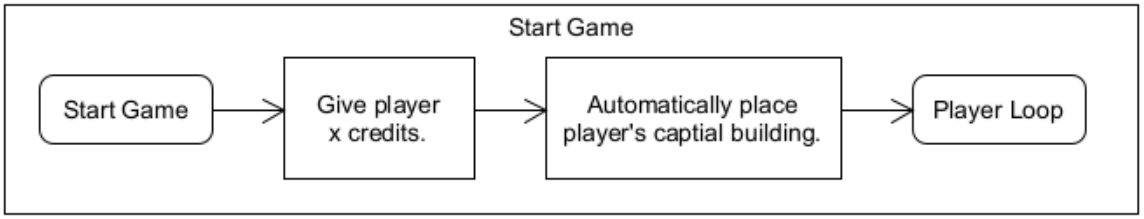
Map UML

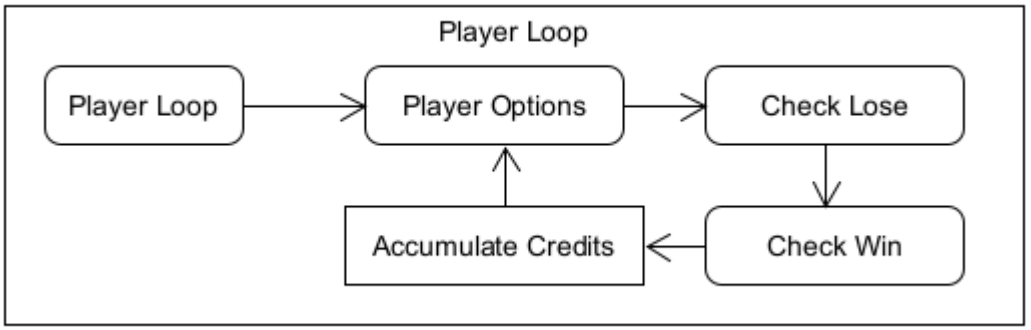

HexagonTileStore
implements ISyncVar
- mTiles : HexagonTile[][]
- mCoordShift : int
- mSeed : int
- mViewedTileMask : Map<Integer, boolean[][]>
- mTileMask : Map<Integer, boolean[][]>
- mTileRowMask : Map<Integer, booleanp[]>
- mDirty : Map<Integer, boolean[]>
- mMap : HexagonMap
- mHander : TileToStoreActions
+ HexagonTileStore(int, int, HexagonMap) : void
+ getTile(int, int) : HexagonMap
+ getAllTiles() : Stream<HexagonTile>
- setTile(HexagonTile) : void
HexagonTile
implements INetSerializable
- mQ : int
- mR : int
- mTileType : TileType
~ HexagonTile(int, int. float, TileToStoreActions) : void
+ length() : int
+ distTo(int, int) : int
+ setClaimedBy(Building) : boolean
+ removeClaim() : void
+ isClaimed() : boolean
+ getClaimant() : Player
+ getClaimedBy() : Building
+ getClaimantId() : Integer
+ setBuilding(Building) : void
+ getBuilding() : Building
+ hasBuilding() : boolean
+ isBuildable(Player) : boolean
uses
uses
uses
HexagonMap
extends NetworkableComponent
implements IOnAwake
- mSize : int
- mTiles : HexagonTileStore
- mLargestLandMass : int[]
- checkIslands() : void
- floodFillLand(HexagonTile) : void
+ floodFill(Deque<HexagonTile>. IFloodFillVisitor, int) : void
+ getTile(int, int) : HexagonTile
+ getAllTiles() : Stream<HexagonTile>
+ getTilesInRadius(HexagonTile, int, boolean, List<HexagonTile>) : List<HexagonTile>
+ getTilesInRadius(HexagonTile, int, int, List<HexagonTile>) : List<HexagonTile>
- getTilesInRadius(int , int , int, int, List<HexagonTile>) : List<HexagonTile>
enum TileType
TYPES:
LAND
WATER
FOG
MOUNTAIN

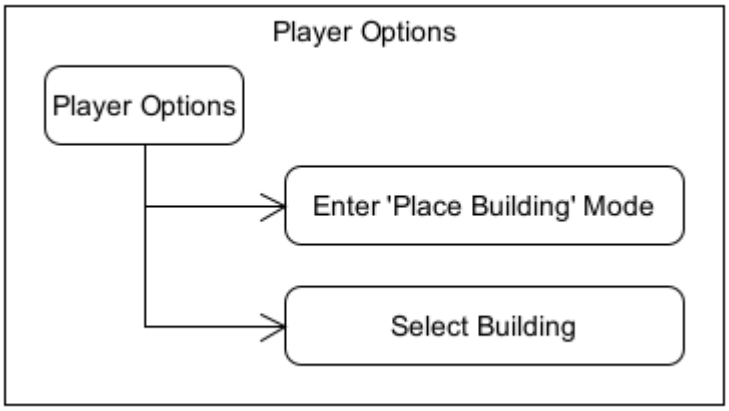
12.3.2 Game Logic Diagrams

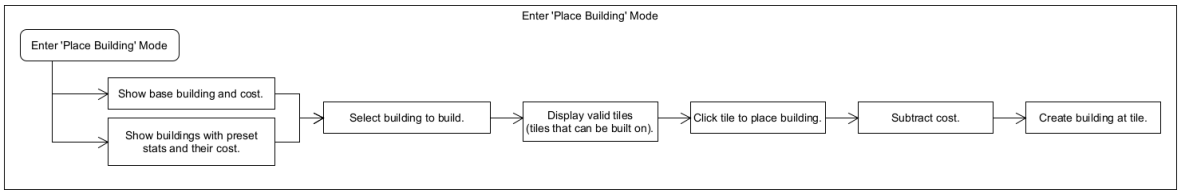
These diagrams were created to allow us to easily visualise the flow of game logic.

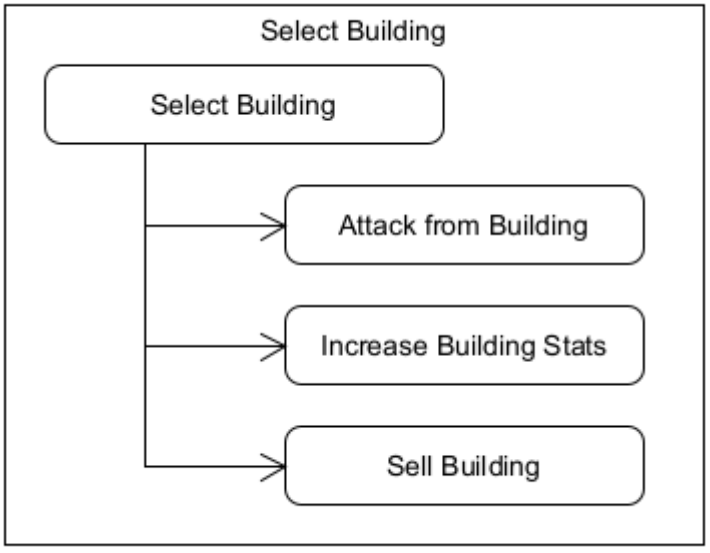


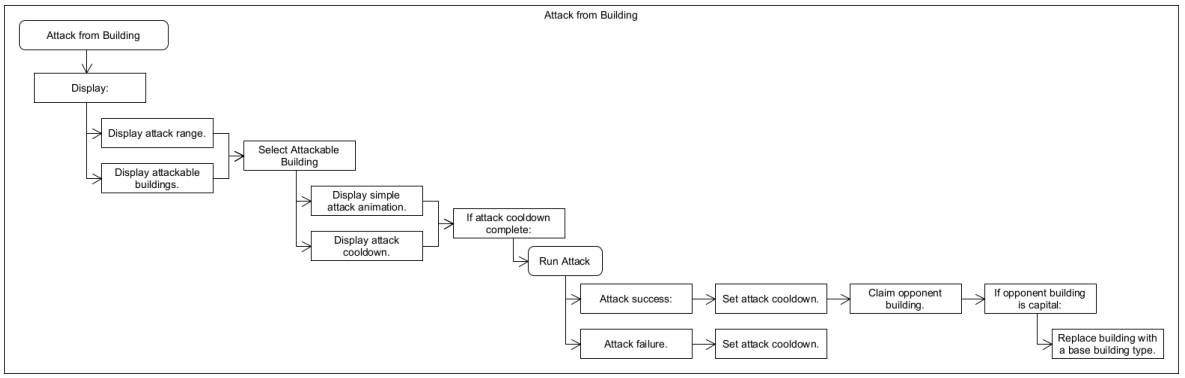


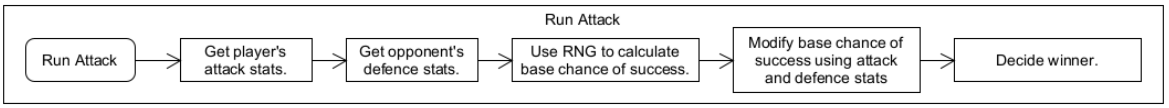


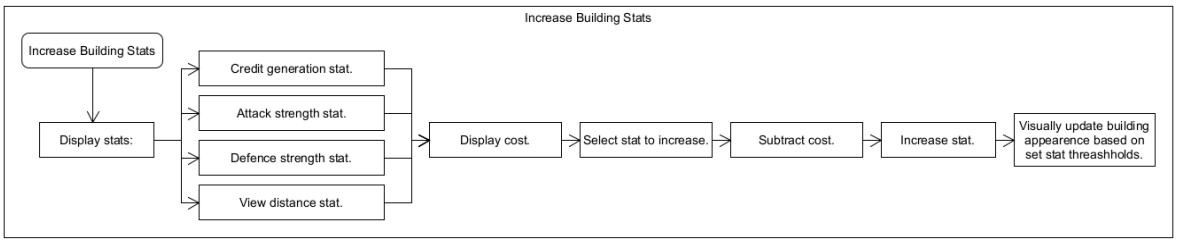

Player Options
Player Options
Enter 'Place Building' Mode
Select Building

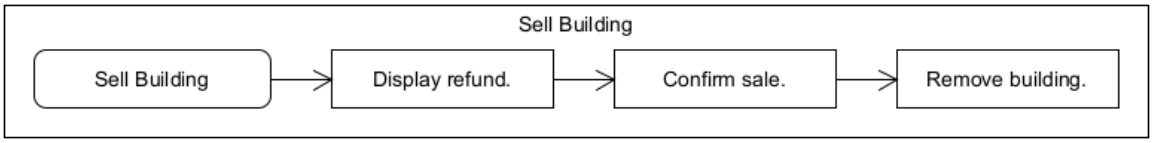

Enter 'Place Building' Mode
Enter 'Place Building' Mode
Show base building and cost.
Show buildings with preset stats and their cost.
Select building to build.
Display valid tiles (tiles that can be built on).
Click tile to place building.
Subtract cost.
Create building at tile.

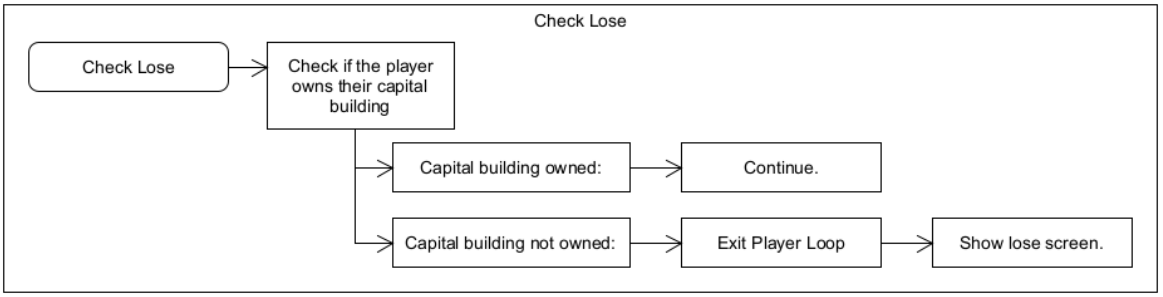

Select Building
Select Building
Attack from Building
Increase Building Stats
Sell Building


Attack from Building
Attack from Building
Display:
Display attack range.
Display attackable buildings.
Select Attackable Building
Display simple attack animation.
Display attack cooldown.
If attack cooldown complete:
Run Attack
Attack success:
Set attack cooldown.
Claim opponent building.
If opponent building is capital:
Replace building with a base building type.
Attack failure.
Set attack cooldown.


Run Attack
Run Attack
Get player's attack stats.
Get opponent's defence stats.
Use RNG to calculate base chance of success.
Modify base chance of success using attack and defence stats
Decide winner.


Increase Building Stats
Increase Building Stats
Display stats:
Credit generation stat.
Attack strength stat.
Defence strength stat.
View distance stat.
Display cost.
Select stat to increase.
Subtract cost.
Increase stat.
Visually update building appearence based on set stat threashholds.


Sell Building
Sell Building
Display refund.
Confirm sale.
Remove building.


Check Lose
Check Lose
Check if the player owns their capital building
Capital building owned:
Continue.
Capital building not owned:
Exit Player Loop
Show lose screen.

13. Test Report

13.1 Testing Strategy

The program has been split into key components; each component has a set of unit tests. Unit tests allow for specific functionality of an individual component to be tested. These unit tests are automated via JUnit. A component must pass all of its unit tests before it is allowed to be integrated into the main program.

System testing will also be completed to ensure the program, as a whole, meets the behaviours defined in the requirements. System testing will be done manually, testing predefined actions and values, and comparing the results against the expected behaviour.

13.2 Principles Used

Early Testing - The team started testing as soon as possible in the Software Development Cycle. In this way we have captured defects in the requirements in an early stage.

Context Dependent Testing - The software is developed in different ways and therefore requires varied testing methods. The software for input testing, for example, is in the form of unit tests. However, the physical implementation of UI required more practical testing, manually checking if the UI provides the correct experience.

Functionality Testing - This testing was done to check if our software met our functional requirements.

Play Testing - Play testing allows us to gain important feedback about the game. This allowed us to understand the users perspective of the game, discover which elements they found more enjoyable-while analysing and optimising parts of the game that were less enjoyable. For example, play testing was crucial to balancing the game and ensuring it provides a consistent user experience.

Performance Testing - This testing was done to check the real time efficiency and performance of the application. Load testing was a part of this testing where we checked if the system was functioning well under load. The general smooth functioning of the game was also tested.

13.3 User Testing

For user testing, we hosted a couple of different sessions in which we asked a variety of users to play the game. During this, we asked them questions about their experiences. We then asked them to fill out a response form, consisting of a checklist and a variety of specific questions, to formally gain their feedback.

The first round of user testing was done before we had finished development. This gave us a good view into the game from an outside perspective, allowing us to fix issues we had missed during our own tests. We used the feedback gained to make improvements- most for visual changes and balance tweaks. These included improvements such as showing a clearer attack animation and ensuring the visual appearance of the buildings properly updated when upgraded.

The second round of user testing was done after development had finished. This allowed us to fully reflect on our finished product, and helped inform us when writing our evaluation (see main report). The feedback we got from the users included the request for tooltips, to help with understanding different actions. Users also said that an in-game tutorial would have been helpful. Furthermore, some users suggested that an AI that more proactively changes its strategy as the game continues. Some users felt that the attacking feature could have been more intuitive, so further modifications to the UI could be made to help clear up any issues.

See 2.1 Results of User Testing.

13.4 Coverage Achieved

Our JUnit tests cover 45%. This appears low due to the size of the program and because many parts of the program, such as rendering and UI implementation, are unable to be properly unit tested as they do not output values which JUnit can read and test (to make up for this, these parts have been covered more by our test plan and play tests).

See 13.6.2 Name of Each JUnit Test Class and 13.6.3 JUnit Test Results (Package Level).

13.5 Test Plan and Results

We initially created the Test Plan (consisting of the Requirements, Priorities and Expected Results). After we finished development, we tested our final product against the Expected Results, filling in the Actual Outcome and specifying where tests passed or failed.

The vast majority of our tests passed. The only failures to occur were ‘Wants’, so not necessary for the game to have.

Some of these tests were minor visual chances, such as adding specialist props for water and mountain tiles. Other tests would have allowed for further customisation, like allowing the host to specify the number of players per game and the map size.

A couple of the tests failed due to changes we made at the end of development, such as not allowing the player to increase view distance and attack range- despite the fact that the logic was fully

implemented. These changes were done to help reduce complexity, as having so many upgrade options harmed the user experience.

This means that the game contains most of our initial requirements, passing all the tests we created in the Test Plan. Furthermore, many of the failed tests are simple additions to the game, which could be done if more time was given.

ID		Requirement	Priority	Expected Result	Actual Result and Comments	Pass / Fail
Map						
1		The system shall display a map formed out of hexagonal tiles.	M	A map, broken up into hexagonal tiles, should be displayed.	A hexagonal map formed out of hexagonal tiles is clearly displayed.	Pass
2	1	The system shall only display visible regions of the map (fog of war).	S	Only a certain radius around owned buildings should be visible.	Only tiles within a certain radius of the player's buildings are visible.	Pass
	2	The system shall update the fog of war when a building is placed.	M	After placing a building, a larger portion of the map should be revealed. The amount of visible map revealed should be dependent on where the building is placed.	Undiscovered areas of the map get revealed when a building is placed, showing the layout of the map in that area and any opponent buildings and claims. Rediscovering areas stops the tiles from being greyed out and once again shows opponents' buildings and claims. The area revealed is in accordance with the view range of the building.	Pass
	3	The system shall update the fog of war when building ownership changes, or the building is destroyed.	C	After losing ownership of a building, the visible area provided by that building should be lost.	When building ownership changes or a building is sold, the area uncovered by the building is greyed out. The player is able to see the layout of the land, but is unable to view any activity there (such as opponent buildings and claims).	Pass
3	1	The system shall utilise a basic map.	M	A map consisting only of land tiles should be used.	This has been surpassed, as the game has been improved from only displaying a basic map of land tiles, to the more complex random map as specified below.	Pass
	2	The system shall generate a random map consisting of different types of hexagon tile.	C	For each new game hosted, a map with newly generated contents should be used.	Every game has a different map consisting of a mixture of land, mountain, and water tiles. These tiles are placed in a random layout each time, based on a coherent noise function.	Pass
4	1	The system shall place a capital building for each player on the map, at a random location.	M	When the game starts, each player should have a capital building automatically placed for	Capitals are spawned at random locations on the map at the start of the game.	Pass

				them somewhere on the map.		
	2	The system shall ensure each player's capital city building is reachable by opponent players.	S	Each player should be able to access every other player.	The game calculates the largest landmass and spawns all capitals on that to guarantee that all players can reach each other.	Pass
	3	The system shall ensure the players' capitals are sufficiently evenly distributed around the map.	S	Each player's capital building should be evenly spaced around the map.	The map is split up into equal segments, with the game attempting to place one capital in each segment. This sufficiently distributes the capitals around the map.	Pass
5		The system shall allow the user to move the map.	S	The user should be able to use keyboard and mouse input to move the map to the desired location.	The map can be moved via keyboard input (using the arrow keys or WASD), and the map can be dragged with the mouse.	Pass
6		The system shall allow the user to zoom in and out of the map.	C	The user should be able to use the mouse scroll wheel to increase and decrease the size of the map.	Zooming in and out can be done via a slider that is displayed on the screen, or with the mouse scroll wheel.	Pass
7		The system shall claim a 1 tile radius by the player around each building.	S	An area around each building should be claimed. This should be visibility distinct to the user, based on their player colour.	Placing a building claims tiles, that have not already been claimed, around it. These claimed tiles are highlighted to show which player they belong to.	Pass
8	1	The system shall have normal land tiles (that can have buildings placed on them).	M	There should be grassland tiles that support buildings being placed on them.	The grassland tiles are clearly visible. These are the only tiles that allow buildings to be placed on them.	Pass
	2	The system shall have water tiles (that cannot have buildings placed on them).	S	There should be water tiles that cannot support buildings being placed on them.	The water tiles are clearly visible. These tiles do not allow buildings to be placed on them.	Pass
	3	The system shall have mountain tiles (that cannot have buildings placed on them.)	C	There should be mountain tiles that cannot support buildings being placed on them.	The mountain tiles are clearly visible. These tiles do not allow buildings to be placed on them.	Pass
9	1	The system shall display a variety of additional 3D props on the land tiles surrounding buildings.	C	A variety of smaller props should appear on the claimed land tiles of each building. These should reflect the current stat levels of the building.	Props are displayed on claimed land tiles. These reflect the current stat levels of the building that claims the tile, the higher the stat, the greater the props shown.	Pass
	2	The system shall display a variety of additional	W	A variety of smaller, specially selected props	Props are not currently displayed on water or mountain tiles.	Fail

		3D props on the non-land tiles surrounding buildings.		should appear on the claimed non-land tiles of each building, for example boats on water tiles. These should reflect the current stat levels of the building.	We decided not to do this due to the large amount of time required to create 3D models , which would only be limited to a few tiles.	
Building Placement						
1		The system shall allow the user to enter ‘Place Building’ mode.	M	The user should be able to select a button to enter building placement mode. When in this mode, the user should be able to select a tile to build there.	When no building is selected, the option to ‘Place Building’ is made available. When selected, all valid tiles are highlighted- as specified below. Selecting a tile causes a building to be placed. Furthermore, the user can press ‘B’ on the keyboard to enter build mode directly.	Pass
2	1	The system shall display the valid tiles that can be built on.	S	The map should clearly highlight tiles that can have buildings placed on them.	The map clearly shows all the valid tiles that can be built on by highlighting them in green.	Pass
	2	The system shall display the tiles that cannot have buildings placed on them.	C	The map should clearly highlight tiles that cannot have buildings placed on them.	The map clearly highlights all the invalid tiles in red. When an invalid tile is selected a place building option does not show up.	Pass
	3	The system shall forbid buildings being placed directly on top of other buildings	M	The option to place a building should not be given if a tile already has a building.	The option to place a building is not given when a tile with a building is selected.	Pass
	4	The system shall forbid buildings being placed outside of the visible region of the map (fog of war).	M	The option to place a building should not be given if a tile outside of the visible area is selected.	The option to place a building is not given when a tile outside the visible region is selected.	Pass
	5	The system shall forbid buildings being placed on any claimed tiles.	M	The option to place a building should not be given if a claimed tile is selected.	The option to place a building is not given when a claimed tile is selected.	Pass
	6	The system shall forbid buildings being placed directly adjacent to each other.	M	The option to place a building should not be given if any of the neighbour tiles, adjacent to to a selected tile, has a building.	The option to place a building is not given when a tile adjacent to a tile with a building is selected.	Pass
3	1	The system shall allow a basic building with no increased stats to be placed.	M	The option to place a building, with all stats all set to the lowest level, should be given.	When a valid tile is selected, the option to place a basic building, with all the stats set at the lowest level, is given.	Pass

	2	The system shall display a shop for placing pre-upgraded buildings.	S	The user should be able to see a shop displayed. They should be able to select the type of building to place from this shop. When they place a building, the building should be upgraded to the selected, preset levels.	A shop is displayed when in Place Building mode. The user can select to place a selection of buildings, each with the stats pre-upgraded to varying amounts. When the building is placed, the building has the selected pre-upgraded stats.	Pass
	3	The systems shall display the cost of placing each building.	S	The user shall see, in the shop, the cost of placing a building.	The cost of placing each building is shown in the shop. The shop also shows if a building is too expensive to be placed by highlighting the cost in red.	Pass
4		The system shall subtract the cost of a building each time that specific building is placed.	M	The amount of tokens a player has should reduce, by the predefined cost, when a building is placed.	The cost of the selected building is subtracted from the player's tokens.	Pass
Selecting a Building (UI)						
1		The system shall allow the user to select a building they own.	M	The user should be able to click a building, and then view the possible options that can be taken with that building.	When a building the player owns is selected, the options to upgrade the building, attack from it and sell it are presented.	Pass
2	1	The system shall provide the player with the option to attack an opponent building.	M	After the user has selected the building to attack from, all targets which could be attacked should be made visible. The user then should be able to select the opponent building they want to attack.	When the player selects to attack from a building they own, all nearby opponent buildings are highlighted in red. The player can then select one of these buildings and launch an attack against it. If no opponent buildings are within range, the player is informed that attacking is not currently possible from the selected building.	Pass
	2	The system shall provide the player with the option to increase the stats of a building.	S	After the user has selected a building, they should be given the option to choose which stats to increase.	When the player selects a building they own, they are shown an upgrade shop. This shows each upgradeable stat, its current level, and the cost to upgrade to the next level.	Pass
	3	The system shall provide the player with the option to sell the building.	C	After the user has selected a building, they should see a button that allows them to sell the building, and displays the potential rebate from doing this.	When the player selects a building they own, they are shown a sell button. Pressing this opens a confirmation menu in which the partial token refund is shown, and a button to confirm the sale is made available.	Pass
Attack an Opponent						
1		The system shall allow	M	Opponent buildings within	Buildings can attack nearby opponent	Pass

		the user to attack opponent buildings.		viewable and attackable distance should be attackable.	buildings. If an opponent building is out of range, they are unable to launch any attacks.	
2		The system shall subtract the cost of the attack from the attacker.	S	The cost of the attack should be subtracted from the attacking player's number of tokens.	When an attack is launched, the cost of the attack is subtracted from the attacking player's tokens.	Pass
3		The system shall have an attack cooldown, so the same player cannot repeatedly spam attacks.	C	Once the attack has been done there should be a cool down period during which the player should not be able to attack again.	After an attack has been launched from any building, there is a short cooldown period during which no more attacks can be done.	Pass
4	1	The system shall highlight buildings that can be attacked (buildings inside the range of attack and that are visible).	S	Visible tiles with opponent buildings on, within attack range, should be distinctly highlighted.	When a building is preparing to launch an attack, attackable opponent buildings are highlighted in red. Furthermore, a projectile path is shown between the attacking building and the attackable buildings, when they are hovered over.	Pass.
	2	The system shall display an attack cooldown.	S	The user should have the attack cooldown clearly displayed when it is active.	The attack cooldown is clearly displayed to the user via a countdown.	Pass.
	3	The system shall display the cost of the attack.	M	The user should see a clear cost for attacking an opponent building.	When a building enters attack mode, its attack cost is clearly shown. Furthermore, if the attack cost is greater than the player's current amount of tokens, the button to enter attack mode shows the desired cost.	Pass.
5	1	The system shall display a simple attack animation to the user and opponent.	C	Both the attacking player and defending player, at a minimum, should see an attack being visibly launched between the respective buildings- regardless of whether the attack succeeds or fails.	An attack animation is clearly displayed with one or more projectiles firing from the attacker to the defender. This is visible to all players.	Pass.
	2	The system shall play an attack audio prompt to the user and opponent.	S	The attacking and defending player should hear an attack launched sound. They should also hear a sound dependent on whether the attack failed or succeeded.	When an attack is launched, both the attacking and defending player hear an attack launch sound. If the attack is successful, the building captured sound is played to both the attacker and defender.	Pass.
6	1	The system shall calculate whether the attack succeeded or failed.	M	Attacks between two players should be able to fail or succeed.	The attacks launched can end in either failure or success.	Pass.

	2	The system shall utilise the player's building attack stat and the opponent building's defence stat to calculate how successful the attack is.	S	The higher the attack stat of a building, the higher the chance the attack should succeed. The defending building should be able to counter this by having the defense stat reduce the chance of success.	The success or failure of an attack depends on the relevant attack and defence stats. When the attacking building has a higher attack stat than the defending building, the probability of success is higher. When the defending building has a higher defence stat than the attacking building, the probability of success is lower.	Pass.
	3	The system will utilise an algorithm that includes random number generation to add a non-deterministic nature to attacks.	M	Attacks should always have a chance to fail or succeed, regardless of how high the stats are.	Attacks always have a chance to fail or succeed, with the scale of this probability being dictated via the buildings' attack and defence stats.	Pass.
7		The system shall award the ownership of the opponent building to the attacker, on a successful attack.	M	On a successful attack, the defending building should change the colour of its claims, and the attacker should get the options to upgrade, sell or launch an attack from the building.	When an attack is successful, the attacker gets ownership of the attacked building- changing the colour of the building's claimed tiles to the player's colour. The player is able to now upgrade, sell or launch an attack from the building.	Pass.
8		The system should allow each building to have a certain amount of health, with failed attacks deducting from this health.	W	On a failed attack, the health of the defending building should be reduced. Buildings with reduced health should have reduced defences- increasing the probability that a future attack will be successful. Buildings should regenerate health over time.	Buildings do not have a concept of health. We decided not to implement building health, but our code is robust enough to allow it in future development. If this was implemented, we would have to ensure that the additional complexity would not harm the user experience.	Fail
Increase Building Stats						
1		The system shall allow the player to increase the stats of their buildings.	S	Each building should have a set of stats, with these stats being able to be individually upgraded by the building's owner.	Each building has a set of stats. These stats are leveled from 1 to 10, and the player has the option to increase the level of a stat, and doing so increases the value of the stat.	Pass
2	1	The system shall allow the player to change their building's stats for a cost.	M	On purchasing an increase in a stat, the cost should be deducted from the player's token amount.	The cost for upgrading the level of a stat is subtracted from the player's token amount.	Pass
	2	The system shall ensure the higher stats a building has, the higher the cost of stats for that	S	The user should see that for each upgrade purchased, the cost visibly increases.	The user can see that each upgrade purchased both increases the cost of that individual upgrade and the cost of upgrading the other stats.	Pass

		building.				
3	1	The system shall allow the player to changet: token generation rate.	S	For each individual building owned by a player, they should be able to increase the token generation rate for that building. Doing so should result in the total amount of tokens for that player generating at a faster rate.	The user is able to upgrade token generation rate. For every level increased, the building produces one more token per token generation cycle.	Pass
	2	The system shall allow the player to changet: attack strength.	S	For each individual building owned by a player, they should be able to increase the attack strength for that building. Doing so should result in attacks having a higher probability of success.	The user is able to upgrade attack strength. For every level increased, the building gains increased attacking power. This is shown via the attack success probability increasing.	Pass
	3	The system shall allow the player to change: defence strength.	S	For each individual building owned by a player, they should be able to increase the defense strength for that building. Doing so should result in defending against attacks having a higher chance of success.	The user is able to upgrade defense strength. For every level increased, the building gains increased defensive power. This makes enemy attacks weaker, reducing the chances of the building being captured.	Pass
	4	The system shall allow the player to change: view distance.	W	For each individual building owned by a player, they should be able to increase the view distance for that building. Doing so should result in the player being able to see a greater area around the building.	Each building does have a view distance stat that determines the view range of a building but, currently, this is set to not be upgradeable. We decided to do this as we found that managing 5 different stats for each building became too complex, so we removed the ability for the view distance to be upgradeable. However, this could be easily reintroduced as it uses the stat code.	Fail
	5	The system shall allow the player to change: attack range.	W	For each individual building owned by a player, they should be able to increase the attack range for that building. Doing so should result in the player being able to attack buildings that are further away, but still within view distance.	Each building does have an attack distance stat that determines the range at which a building can attack opponents but, currently, this is set to not be upgradable. We decided to do this as we found that managing 5 different stats for each building became too complex, so we removed the ability for the attack distance to be upgradeable. However, this could be easily reintroduced as it uses the stat code.	Fail

					Furthermore, we found that additional complexity was added by attack range due to the fact that only visible buildings can be attacked- so the attack range always had to be lower or equal to the view distance to have any impact.	
4		The system shall change the type of building displayed based on the stats of a building.	C	The user should see the building change its 3D model dependent on which of the stats is upgraded to the highest amount.	The user can see the building change its 3D model based on which stat is upgraded to the highest amount, unless the building is the capital. Furthermore, the higher level the stat, the more the 3D model changes.	Pass.
Selling a Building						
1		The system shall allow for buildings to be sold for a partial refund.	C	When selected, buildings should have the option to be sold and removed from the map. Doing so should refund some tokens.	When selected, a building displays a sell option. Pressing this asks for the user's confirmation; doing so removes the building from the map and provides some tokens, at a lower rate than what the building was purchased for.	Pass
2		The system shall remove the building being sold.	M	When a building is sold, it should be removed from the game map.	When a building is sold, it is removed from the map.	Pass
Token Generation						
1	1	The system shall allow for each player to generate a certain amount of tokens every few seconds.	M	A player's total amount of tokens should slowly increase over time.	A base rate of tokens are added roughly every second.	Pass
	2	The system shall provide the number of tokens based on the token generation rate of that player's buildings.	S	The higher the token generation rate each player's buildings, the more tokens should generate.	The more buildings a player has, and the higher their token generation rates are upgraded, the more tokens are generated.	Pass
Losing a Game						
1	1	The system shall cause a player to lose if their capital building is lost.	M	The player should be displayed a loss message once their capital has been lost.	After losing their capital, a player is shown a loss message and sound effect.	Pass
	2	The system shall prevent a player who has lost from performing any actions that affect gameplay (placing	M	The player should be unable to perform any actions after losing. Users, however, should still be able to spectate the map-	Players who have lost cannot perform actions like building, upgrading, attacking or selling. Players are given the option to spectate the match, allowing them to see the entire map.	Pass

		buildings, attacking opponent buildings or increasing building stats).		but should be unable to interact with it.		
	3	The system shall allow the user to leave the game once they have lost.	C	The user should be able to press a button to take them to the main menu.	The user is able to leave the game straight away after losing, or at any time while spectating via the pause menu.	Pass
2		The system shall remove all of the losing player's buildings.	C	A player should have all buildings owned by them, at the point of loss, removed from the game map.	Once a player has lost, any buildings- that have not already been claimed by others- are destroyed and removed from the map.	Pass
Winning a Game						
1	1	The system shall end the game when all capital buildings have been destroyed, except for the winning player's capital building.	M	At the end of the game, the winning player should be informed of their success.	If a player is the last player with a capital building, they are shown a win message and a sound effect is played.	Pass
	2	The system shall display a victory message to the winning player.	M	The user will see a small message informing them that they have won.	The winning player sees a victory message, and like a losing player, they are given the option to spectate the map or quit.	Pass
	3	The system shall allow the user to leave the game once they have won.	S	The user should be able to press a button to take them to the main menu.	The user is able to leave the game straight away after winning, or at any time while spectating via the pause menu.	Pass
AI						
1		The system shall allow for simple AI players that can perform the same actions as human players (place buildings, attack opponent buildings and increase building stats).	M	Players not being controlled by human players should exist. These AI opponents should not be obviously distinguishable as they should have the same abilities as a human player.	AI opponents can be spawned in. These have the exact same abilities as human players, and follow the exact same validity checks and conditions.	Pass
2		The system shall have AI players that act probabilistically. These will perform different actions based on predefined probabilities.	M	AI players should act probabilistically- doing a mixture of placing buildings, attacking opponents and upgrading- all at a predefined rate.	AI players can build, attack, upgrade and sell. The probability of doing any of these actions can be predefined, allowing the AI to be modified to act in certain ways.	Pass
3		The system shall have AI players that use pathfinding to plan	C	AI players should be able to utilise pathfinding to find the shortest path to a	AI players can target visible opponent buildings by using an A* search to find the shortest path to the target building. The AI	Pass

		routes to attack opponent players.		target building. They should then be able to enact this plan, placing buildings and attacking until they reach the target point.	then will place buildings and attack until it reaches and captures the target building, or until it decides to target a different visible opponent building.	
4		The system shall allow AI players to act both probabilistically and utilise pathfinding.	C	AI players should randomly vary between specific targeting of opponents, and general probabilistic expansion.	AI players act probabilistically until they meet an opponent. They then pick an opponent building to target. Each turn the AI has, it either will act probabilistically or build and attack towards its target.	Pass
5		The system shall ensure AI players prioritise attacking opponent capitals.	S	When opponent capitals appear in the AI player's visible area, they should be more focused on attacking the capitals and surrounding buildings.	If an AI player selects an opponent to target with pathfinding, and the opponent's capital is within its view range, it will always choose to set the capital as the target for the A* search. This means that the AI focuses on defeating that player when doing the specific targeting.	Pass
6		The system shall allow AI players to become more focused on attacking opponent capitals as the game continues.	W	AI players should have a higher chance of targeting opponent capitals, regardless of whether they are within the AI's immediate viewable range, the longer the game goes on for.	Each time the AI player picks a new opponent building to specifically target, it has a chance to select the opponent's capital, even if it is not in visible range. The probability of this occurring increases a small amount every time a new building is specifically targeted.	Pass
7		The system shall allow only the server to control AI players.	M	The server, and not the clients, should have sole control over AI players.	The AI is controlled by the server.	Pass
Menus (UI)						
1	1	The system shall display a start menu.	M	The user should be able to see a main menu when they load up the game.	The main menu is displayed, allowing the user to play the game, access the settings, or exit.	Pass
	2	The system shall display an option to play the game.	M	The user should be able to interact with a button to play the game.	A button, that when interacted with, allows the user to either host or join a game.	Pass
	3	The system shall display an option to edit sound settings.	C	The user should be able to interact with a button that opens sound settings. From this, they should be able to mute audio completely, or change the volume.	A settings button is displayed, which allows access to the audio settings.. The player can increase/decrease the audio volume via a slider, or use a button to mute/unmute audio.	Pass
2	1	The system shall display an options menu when in game.	S	This should act similarly to a traditional pause menu, giving various options related to playing the	A pause menu is available when the user presses ESC. From this they can resume the game, access audio and visual settings, and quit the game.	Pass

				game, without actually pausing the game.		
	2	The system shall prevent the user who opened the options menu from interacting with the game while the menu is open.	M	The user should be unable to interact with the map or buildings while the menu is open. They should only be able to interact with the game when the menu has been closed.	The player is not able to interact with any element of the game when the pause menu is opened. When the options menu is closed, interacting with the game is possible again.	Pass
	3	The system shall display an option to resume playing the game.	M	This option should allow the user to resume playing the game.	An option to resume the game is available when the player has the pause menu open.	Pass
	4	The system shall display an option to edit settings.	C	The user should be able to access the settings.	When the pause menu is open, the user is able to access audio and visual settings.	Pass
	5	The system shall display an option to quit the game.	S	The user should be able to exit a game in progress.	The pause menu shows a button that can quit the game.	Pass
3	1	The system shall provide an interface to upgrade a building's stats.	S	The user should be able to view a stats shop, for each building, that displays the stats that can be upgraded, their current levels, and the cost to upgrade each stats.	When a valid building is selected, the player is shown a stats shop. This displays the current level of each stat, and the cost to upgrade each stat.	Pass
	2	The system shall provide an interface for selecting a building type to place.	S	The user should be able to view a building placement shop in which different types of pre-upgraded buildings are made available to them. They should be able to select a building type, and then select a tile to place it on.	The player is able to see a building shop when in building mode. Each type of building displays its name, pre-set stats and cost. When the player selects a tile, the specified type of building is built.	Pass
	3	The system shall provide an interface for attacking a building.	S	The user should be able to see a menu that allows them to launch an attack from a selected building against an opponent building. This should display the cost of an attack, whether the attack is in countdown, and the probability of success.	The player is able to select a building to launch an attack from. This displays the cost of an attack and, when an opponent is selected, the probability of success. If the player is in cooldown, or that building is being attacked, the time until they can launch an attack is clearly shown via a countdown. Furthermore, a projectile path is shown to clearly display which building the attack and defence is between.	Pass
Rendering						
1		The system shall allow	C	The game map should be	The player can see the game represented in	Pass

		for perspective rendering.		represented in 3D.	3D.	
2	1	The system shall allow for game objects to be represented by basic 3D shapes.	M	Simple 3D shapes should be displayable.	The player can see all the game objects in 3D.	Pass
	2	The system shall allow for game objects to be represented by complex 3D meshes.	C	More complex meshes should be displayable.	The player can also see some game objects represented in complex 3D models. Furthermore, a building's 3D model gets more advanced the higher the level of the building's stats.	Pass
3		The system shall allow for text to be displayed.	S	Text should be displayable on screen.	The player can see text being displayed in the game.	Pass
4		The system shall use physically based rendering.	C	Lighting should be rendered.	The player can see a sun, which moves through the sky, lighting the map, in the game.	Pass
Networking						
1	1	The system shall allow the user to host a multiplayer game.	M	The option to host a multiplayer game should be given.	The user can select the 'Host Game' button. This then presents the option to host on localhost, or over the internet. They are then able to start the game whenever they choose; this will start the game for all players, while also allowing the host to play.	Pass
	2	The system shall allow the user to specify the number of players before a game starts.	W	The host should be given the option to change the number of players each game supports.	We currently do not allow the user to change the number of players per game, due to time constraints. However, could be simply implemented as the number of players per game is already stored in a variable, and it would just be a case of modifying this value via the UI.	Fail
	3	The system shall allow the user to specify the size of the map before the game starts.	W	The host should be given the option to specify the size of the map.	We currently do not allow the user to change the size of the map, due to time constraints. However, most of the internal functionality to allow this does exist, with it being a case of adding a suitable UI for the size selection.	Fail
2		The system shall allow the user to join a multiplayer game.	M	The option to join a multiplayer game should be given.	The user can select the 'Join Game' button. This then presents the option to join on localhost, or join a game being hosted over the internet. Doing so makes them wait for the host to start the game, and connects	Pass

				them to the game when it begins.	
3	The system shall allow the user to select a game to join from a list of available hosts.	W	A list of hosted games, waiting to start, should be displayed. Selecting a game should allow you to connect to the host and play the game.	The user can select a game from a list of hosted games which are waiting to start. When a game is selected the user connects to the host and can play the game.	Pass.
4	The system shall fill a game with AI players, if there are not enough human players.	M	When a game is started, any remaining players should be controlled by AI.	Each game can have up to 6 human players. If these are not filled by humans on game start, AI players will be spawned in to make up the numbers.	Pass
5	The system shall disconnect the clients if the host ends the game.	S	When a user stops hosting, all clients should be disconnected.	When a host exits the game, all clients are disconnected.	Pass
6	The server shall manage the authoritative game state.	M	The server should handle the game, and clients should use copies that get updated by the server.	The server holds the authoritative game state. Modifying a client does not impact the server or other clients.	Pass
Audio					
1	The system shall allow for background music to be played.	M	The user should be able to hear looped background music while in the menus and playing the game.	Looped background music is played.	Pass
2 1	The system shall allow for spot effects to be played.	M	The user should be able to hear audio that does not loop.	Spot effects are able to be played.	Pass
2 2	The system shall allow for multiple spot effects to be played simultaneously.	S	Multiple sound effects should be able to be played at once.	Up to 32 different sound effects can be played simultaneously.	Pass
2 3	The system shall play spot effects when key actions occur.	M	The user should be able to hear sound effects when they undertake key actions, from pressing menu buttons, to placing buildings.	Sound effects are played at key actions, including: on UI button presses, at game start, at game end, when attacking, when defending, when building, when upgrading, and when selling.	Pass
3 1	The system shall allow audio to be muted.	M	The option to silence audio should be given.	The user can enter the settings menu, and from there adjust audio options- including the option to mute audio.	Pass
3 2	The system shall allow for audio volume to be adjusted.	C	The option to change the volume of audio should be given.	The user can enter the settings menu, and from there adjust audio options- including a slider to change the volume of audio.	Pass
4	The system shall	W	Audio options should be	Audio settings are stored in a settings file,	Pass

		preserve audio options across multiple launches of the game.		preserved when the game is closed and relaunched.	allowing for persistent audio settings.	
5		The system shall fade volume by distance from the location of the spot effect.	W	The further away a sound effect occurred, the quieter the sound should play.	Spot effects can be played with buildings as their source, so the further away the building, the quieter the sound effect.	Pass
Input						
1		The system shall accept keyboard input.	M	The user should be able to use the keyboard to control the camera and pause the game.	The user is able to control the camera using the keyboard (via both the arrow keys and WASD). The user can also press ESC to open and close the pause menu, as well as traverse out of the pause menu. Furthermore, keyboard shortcuts to place buildings, attacking and selling have been added.	Pass
2	1	The system shall accept mouse button input.	M	The user should be able to use the mouse buttons to interact with menus and select map tiles.	The user is able to both select tiles and interact with buttons and sliders, by clicking on them.	Pass
	2	The system shall allow for mouse scrolling to be detected.	C	The user should be able to use the mouse scroll wheel to zoom the map in and out.	The user is able to zoom in and out of the map by using the scroll wheel.	Pass
3	1	The system shall track mouse position.	M	The user should be able to see their cursor location, and interact with menus and the map at that location.	A custom mouse cursor is displayed, allowing the player to easily see their cursor location. They are able to use the mouse to interact with menus and the map.	Pass
	2	The system shall allow for mouse dragging to be detected.	C	The user should be able to drag the mouse to move the map.	The user can click and drag on the map to move the camera.	Pass

14 Test Report Appendix

14.1 User Testing Form

Test	**Success (Y/N)**
Menu	
A menu is displayed allowing you to host or join a game.	
A menu is displayed allowing you to play the game either locally or publically.	
A menu is displayed allowing you to change audio and graphics settings.	
Map	
You see the map as soon as the game loads.	
You can only see the area of the map around your buildings.	
Tokens	
You can see the number of tokens you have.	
Placing Buildings	
You understand how to place a basic building.	
You understand how to place more advanced buildings.	
You understand where on the map buildings can be placed.	
Upgrading Buildings	
You understand how to access the menu to upgrade a building’s stats.	
You understand how to upgrade your attack.	
You understand the impact that upgrading attack has on the game.	
You understand how to upgrade your defence.	
You understand the impact that upgrading defence has on the game.	
You understand how to upgrade your token generation.	
You understand the impact that upgrading token generation has on the game.	
You understand the costs associated with upgrading.	
Sell Building	
You understand how to sell buildings.	

You understand the refund you get for selling.	
Attack	
You understand how to attack an opponent.	
You understand the costs associated with attacking.	
You understand the cooldown.	
You understand when you have lost an attack.	
You understand when you have won an attack.	
You understand when you are being attacked.	
AI	
The AI opponents provide a challenge.	
The AI opponents aren’t impossible to defeat.	
End Game	
You understand when you win/lose.	
You understand what causes a win or loss.	

14.2 User Testing Responses

14.2.1

Test	**Success (Y/N)**
Menu	
A menu is displayed allowing you to host or join a game.	Y
A menu is displayed allowing you to play the game either locally or publically.	Y
A menu is displayed allowing you to change audio and graphics settings.	Y
Map	
You see the map as soon as the game loads.	Y
You can only see the area of the map around your buildings.	Y
Tokens	
You can see the number of tokens you have.	Y
Placing Buildings	
You understand how to place a basic building.	Y
You understand how to place more advanced buildings.	Y
You understand where on the map buildings can be placed.	Y
Upgrading Buildings	
You understand how to access the menu to upgrade a building's stats.	Y
You understand how to upgrade your attack.	Y
You understand the impact that upgrading attack has on the game.	Y
You understand how to upgrade your defence.	Y
You understand the impact that upgrading defence has on the game.	Y
You understand how to upgrade your token generation.	Y
You understand the impact that upgrading token generation has on the game.	Y
You understand the costs associated with upgrading.	Y
Sell Building	
You understand how to sell buildings.	Y

You understand the refund you get for selling.	Y
Attack	
You understand how to attack an opponent.	Y
You understand the costs associated with attacking.	N
You understand the cooldown.	Y
You understand when you have lost an attack.	Y
You understand when you have won an attack.	Y
You understand when you are being attacked.	Y
AI	
The AI opponents provide a challenge.	Y
The AI opponents aren't impossible to defeat.	Y
End Game	
You understand when you win/lose.	Y
You understand what causes a win or loss.	Y

How complete does the game feel *(0 - very bad, 5 - very good)* ?
3.5

How much does the design of the game appeal to you *(0 - very bad, 5 - very good)* ?
4

How intuitive is the UI to use *(0 - very bad, 5 - very good)* ?
3.5 - It works and I understood how to play, but the attacking wasn't very obvious.
Maybe a coins per second stats on the screen would be a nice feature.

What else would you like to see in the game?
Improved UI for the attacking.
Tooltips or similar to explain buildings better.
Maybe icons for different buildings.
Maybe different races or clans to select at start with different bonuses.

How enjoyable was the game?
Fun! Definitely has lots of potential.

How much would you pay for it?
£5 now as an alpha, £15 if it was fully working and finished.

14.2.2

Test	Success (Y/N)
Menu	
A menu is displayed allowing you to host or join a game.	Y
A menu is displayed allowing you to play the game either locally or publically.	Y
A menu is displayed allowing you to change audio and graphics settings.	Y
Map	
You see the map as soon as the game loads.	Y
You can only see the area of the map around your buildings.	Y
Tokens	
You can see the number of tokens you have.	Y
Placing Buildings	
You understand how to place a basic building.	Y
You understand how to place more advanced buildings.	Y
You understand where on the map buildings can be placed.	Y
Upgrading Buildings	
You understand how to access the menu to upgrade a building's stats.	Y
You understand how to upgrade your attack.	Y
You understand the impact that upgrading attack has on the game.	Y
You understand how to upgrade your defence.	Y
You understand the impact that upgrading defence has on the game.	Y
You understand how to upgrade your token generation.	Y
You understand the impact that upgrading token generation has on the game.	Y
You understand the costs associated with upgrading.	Y
Sell Building	
You understand how to sell buildings.	Y
You understand the refund you get for selling.	Y

Attack	
You understand how to attack an opponent.	Y
You understand the costs associated with attacking.	Y
You understand the cooldown.	Y
You understand when you have lost an attack.	Y
You understand when you have won an attack.	N
You understand when you are being attacked.	Y
AI	
The AI opponents provide a challenge.	N
The AI opponents aren't impossible to defeat.	Y
End Game	
You understand when you win/lose.	Y
You understand what causes a win or loss.	Y

How complete does the game feel *(0 - very bad, 5 - very good)* ?
3 - It's very fun to play, functions pretty well. However a few bugs/features make it slightly difficult to play (e.g. if a settlement is built so that it occupies only one hex and none of the surroundings, it can be hard to see which player it belongs to in the heat of a fight).

How much does the design of the game appeal to you *(0 - very bad, 5 - very good)* ?
5 - simple but very effective, absolutely no visual clutter, and I'm a fan of tile-based games anyway.

How intuitive is the UI to use *(0 - very bad, 5 - very good)* ?
4 - could do with some pointers on how to do certain things but nicely designed.

What else would you like to see in the game?
Slightly more aggressive AI, a more defined beginning/middle/end-game, some way to see the controls/hotkeys, less ability to spam click attacking/ building, a way to tell the difference between a town/city/guild (variety is nice!), maybe an overall empire upgrade tree? (i.e. specialise in attack or defense for all buildings?), different types of empire, special attacks (like a culture bomb that takes over many more tiles), and the ability to change sound while in the game lobby.

How enjoyable was the game?
Really fun, although a large part of that was being on a call with friends you are playing with. It seemed fairly well balanced, and the player actually has the potential to go deeper into the stats to find the best strategy.

How much would you pay for it?
In its current state, not much (less than a fiver). But it has potential, and if it had loads more features and the design was a bit more fleshed out, I would pay up to £10, as long as my friends also bought it.

14.3.3

Test	**Success (Y/N)**
Menu	
A menu is displayed allowing you to host or join a game.	Y
A menu is displayed allowing you to play the game either locally or publically.	Y
A menu is displayed allowing you to change audio and graphics settings.	Y
Map	
You see the map as soon as the game loads.	Y
You can only see the area of the map around your buildings.	Y
Tokens	
You can see the number of tokens you have.	Y
Placing Buildings	
You understand how to place a basic building.	Y
You understand how to place more advanced buildings.	Y
You understand where on the map buildings can be placed.	Y
Upgrading Buildings	
You understand how to access the menu to upgrade a building's stats.	N
You understand how to upgrade your attack.	N
You understand the impact that upgrading attack has on the game.	N
You understand how to upgrade your defence.	Y
You understand the impact that upgrading defence has on the game.	Y
You understand how to upgrade your token generation.	N
You understand the impact that upgrading token generation has on the game.	N
You understand the costs associated with upgrading.	Y
Sell Building	
You understand how to sell buildings.	N
You understand the refund you get for selling.	N
Attack	

You understand how to attack an opponent.	Y
You understand the costs associated with attacking.	N
You understand the cooldown.	N
You understand when you have lost an attack.	Y
You understand when you have won an attack.	Y
You understand when you are being attacked.	Y
AI	
The AI opponents provide a challenge.	Y
The AI opponents aren't impossible to defeat.	Y
End Game	
You understand when you win/lose.	Y
You understand what causes a win or loss.	Y

How complete does the game feel *(0 - very bad, 5 - very good)* ?
4

How much does the design of the game appeal to you *(0 - very bad, 5 - very good)*?
5

How intuitive is the UI to use *(0 - very bad, 5 - very good)*?
3

What else would you like to see in the game?
A tutorial mode as it was quite hard to know the objective of the game or how to play it at first. This could even just be a text bubble/annotations.

How enjoyable was the game?
4

How much would you pay for it?
£5

14.2 Name of Each JUnit Test Class

JUnit Class List Engine

- GLTFTest
- DiffBlueWaveSoundTest
- DiffBlueTransform3DTest
- TransformHexTest
- DiffBlueResourceTest
- DiffBlueSceneTest
- GameObjectTest
- ReferenceTest
- ResourceManagerTest
- InputTest
- ScrollTest
- TestActions
- TestBindings
- DiffBlueNetworkClientTest
- DiffBlueNetworkMessageTest
- NetworkedTransformTestComponent
- ServerTest
- TestAttackData
- TestNetworkComponent
- AppTest
- ShaderBufTest
- DiffBlueEnvTest
- DiffBlueTextUtils

JUnit Class List Game

- AppTest
- AttackTest
- CostTest
- DiffBlueNoiseUtilTest
- MaterialsTest
- UpgradeTest
- DiffBlueBuildingTest
- DiffBlueStatTypeTest
- DiffBlueSyncStatTest
- MapTest
- PlayerTest
- AStarTest
- DiffBlueGraphTest
- DiffBlueNodeTest

14.3 JUnit Test Results (Package Level)

Package:	Tests	Errors	Failures	Skipped	Success Rate	Time
org.dragonskulle.components	27	0	0	0	100%	0.01
org.dragonskulle.core	40	0	0	0	100%	0.02
org.dragonskulle.input	17	0	0	0	100%	0.131
org.dragonskulle.network	13	0	0	0	100%	3.754
org.dragonskulle.renderer	7	0	0	0	100%	0.011
org.dragonskulle.assets	8	0	0	0	100%	0.607
org.dragonskulle.audio.formats	1	0	0	0	100%	0.014
org.dragonskulle.utils	6	0	0	0	100%	0.005
org.dragonskulle.game.player.ai.algorithms.graphs	4	0	0	0	100%	0.01
org.dragonskulle.game.player	13	0	0	0	100%	0.785
org.dragonskulle.game.map	1	0	0	0	100%	0.323
org.dragonskulle.game.building.stat	11	0	0	0	100%	0.019
org.dragonskulle.game.building	7	0	0	0	100%	0.034
org.dragonskulle.game	16	0	0	0	100%	2.133