

Pillage the Kingdom

Technical Design Document

Platform(s):

PC

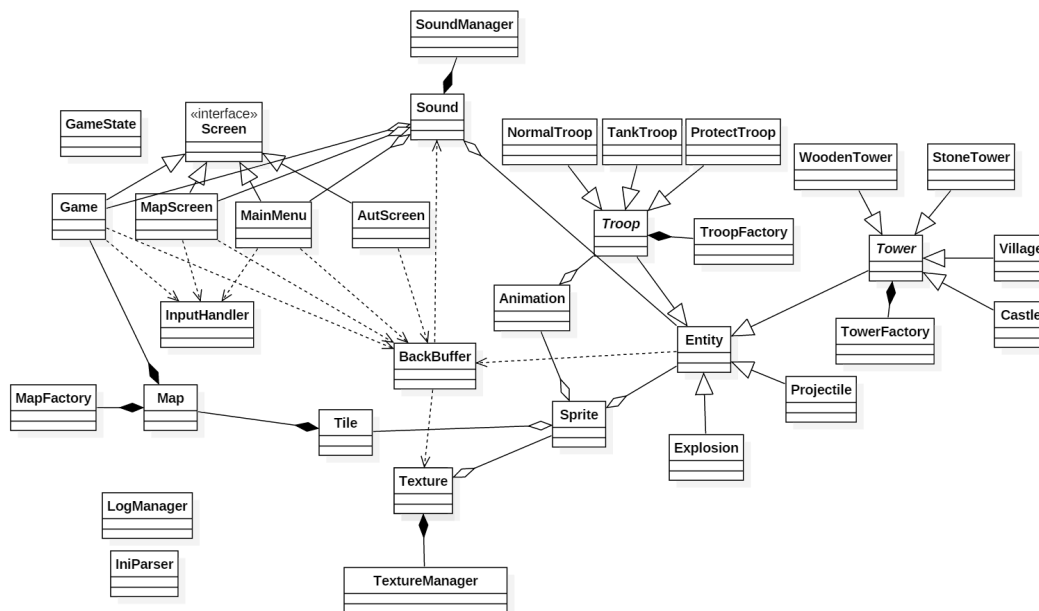
Development Environment and Tools:

Microsoft Visual studio 2013, SDL, FMOD, SVN, Lua

Key Technical Challenges:

Balancing the difficulty of the game. We have to make sure the game is adequately difficult enough for the game to be engaging. But it also has to be easy enough so that it's possible. The possible solution for this challenge would be to playtest the game ourselves and with other people. Using the feedback from this, we can change values around for the gameplay and make the game appropriately more difficult/easy.

Architecture Overview:



Design:

Screens

Within the main, there would be a stack of screen implementations. Each implementation of the screen interface ('Game', 'MainMenu', 'AutScreen' and 'MapScreen') is also singleton design. This would mean that upon a state change within the 'GameState', the main would change focus on which singleton is used.

Troops

Troops are created from a singleton factory ('TroopFactory') and are placed into a building queue. The building queue where they wait until their build time elapses, once it elapses the Troop is transferred to the track queue. Each individual troop is state designed.

Towers

Towers are created from a singleton factory ('TowerFactory') and are placed onto the vector. This vector is with the 'game' class, except for the Village/Castle creations which will be directly referenced. Each individual tower is a state design.

Map

Maps are created from a singleton factory ('MapFactory') and are used to generate a map for the use of 'game'. The 'game' initializers should take in a 'backbuffer' and a 'map'. Each map contains a vector of sprites (art flairs) and a 2D array of 'tiles'.

Managers/Parsers

Managers and parsers are a singleton which are used to open an ini file, output log information, create a texture or create a sound. The 'TextureManager' and 'SoundManager' also restrict the creation of assets, so that duplicates aren't created.

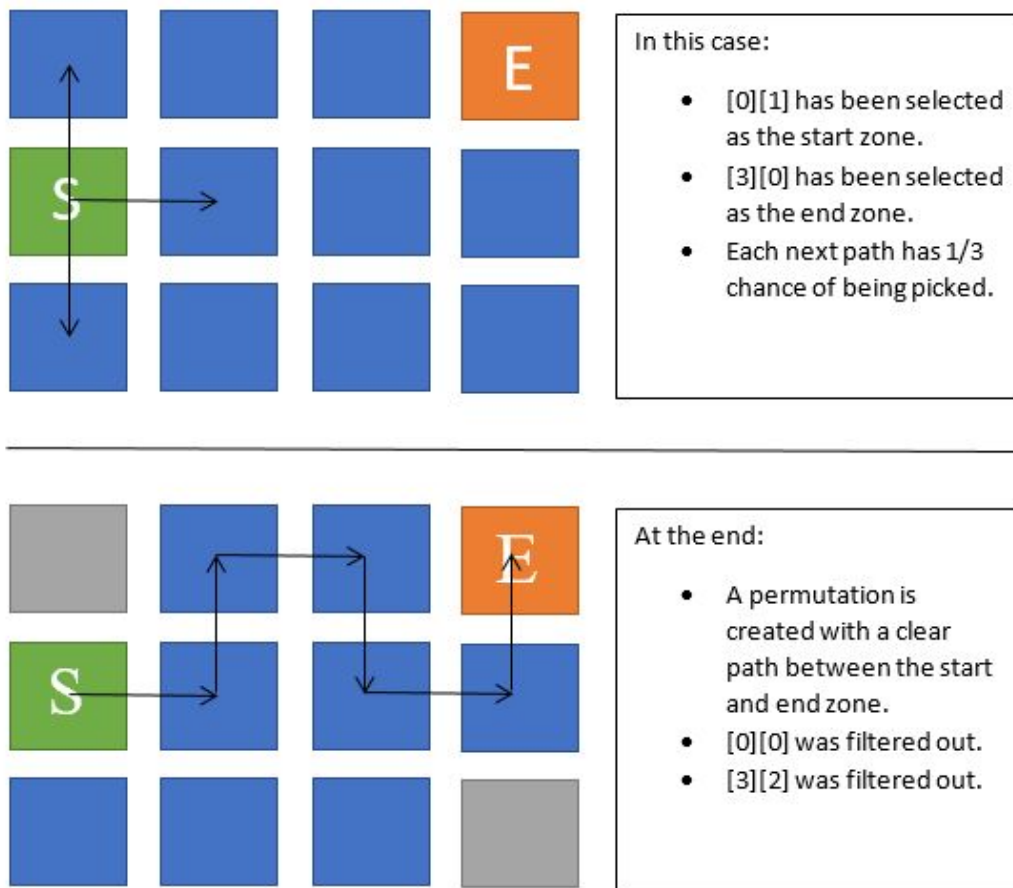
Key Algorithms:

Map Generation Algorithm

The mapping algorithm comprises of 3 coding stages: Pathing, Sampling and Drafting.

- Pathing Stage:

The pathing section starts with two randomly selected tiles. One on the left most side, is designated the starting zone and one on the right most side is designated the end zone. Then the computer will randomly select a path through the other tiles until it reaches the end zone. This method is inspired by Spelunky and its cave design (Yu, 2016).



The problems with this stage is that there are tiles which need to be filtered out. Before a tile is randomly selected, each tile option is assessed. If a tile breaks the track by not reaching the end or being already apart of the track, it is removed.

- Sampling Stage

Only three styles of tiles need to be created; **Blank path**, **Straight path**, **Turn path**. Each tile is randomly selected depending on which tile is required.

Blank Path:

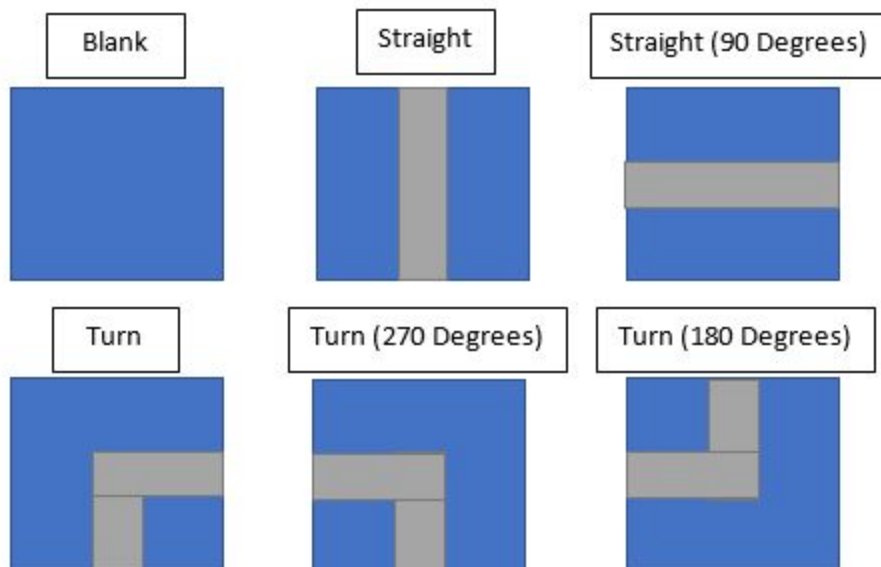
These tile sprites don't have a visible path on them. They are to be used on the sections that aren't connected to the track.

Straight Path:

These tile sprites have one entry and one exit on the exact opposite side. These tiles can be rotated to 0 degrees (north/south) and 90 degrees (west/east) depending on the path required.

Turn Path:

These tile sprites are used within the corners of the track, and are rotated 0/90/180/270 degrees depending on the turn required.



- Drafting Stage:

The final stage to add flair and towers to the map. Towers are placed on the track, while additional flairs like shrubs, trees, etc, are added onto the blank tiles. The tower quantity would be decided by a difficulty level scale (which is predetermined).

- Additional notes:

Within the tile class, there will be three coordinate values: entry point, middle point and exit point. These are based on the assumption that only one axis will be manipulated at a time (No diagonal movement). These coordinates would be initialized and passed through upon request from a troop.

Tower targeting algorithm

The tower targeting algorithm can be broken down into two stages: scanning, firing.

Scanning

The scan stage is where the tower reads through all troops and checks if their coordinate is within range of the tower. If the troop is within range, it is then compared to the current target's focus value. The focus is a value which is calculated with the following values: how far the troop is towards the tower and the troop type value within the ini file. If the new troop has a higher focus value then it replaces it.

Firing

Once a delay has been elapsed the tower will create a bullet, passing through the current target, in which the bullet will LERP towards the designated target. Note: The bullet's target should never change, even if the tower has changed targets.

Troop pathing AI algorithm

The troop has three AI states:

- Moving: The default state, where the troop moves through the map.
- Acting: A conditional state, which enacts the special characteristic of the troop. For example: Protection troops would protect and the tank/normal troop would attack.
- Halted: A triggerable state, which stops the troops current action when the player presses the spacebar.

Note: In the event of units colliding, the collision will only allow the acting/halted state. If they request to move, only the one higher up in the queue is allowed to move while the others stay halted.

Development Methodology:

SCRUM: The reasoning behind this is all group members have experience using SCRUM. It is an easy way to distribute tasks among the groups. Our artefacts are all due weekly so we will be having weekly sprints. It is easy to rank user stories based on difficulty and importance using SCRUM as well. We will be using Trello to track progress on the project.

Trello Link:

<https://trello.com/invite/b/li28yW3T/1b1cf8c4883f92f081337143600cc84b/pillage-the-kingdom>

Risk Management:

Risk: Team Member does not attend planned meeting

Assessment: This could mean that the absent team member will fall behind and not understand what progression has been made in the missed session. Could also mean that the team has made decisions that aren't necessarily something that the absent team member will sign off on.

Response: Make ruleset of what a team member should do if they cannot attend a meeting, such as let the rest of the team know of your absence, and be able to still connect with the team remotely (skype, discord, etc) so that you can be a part of the meeting, just from another location.

Risk: Estimation of size of project we can tackle in the time frame is too large.

Assessment: This could mean that down the line, we have too much to do in the little amount of time we have to do it. Having a huge scope to try and achieve can cause stress/emergency replanning that won't be thought through properly.

Response: Look at aspects of the project that are easily scalable, and plan out a minimum and a maximum scope. Aim for the maximum, and if the team agrees that it is too much work to complete before the deadline, we refer to the scope scale, and reduce the scope to a point where we think we can complete comfortably.

Risk: Features can be much more difficult/involve a lot more time than initially thought

Assessment: When a team member starts to develop a particular feature, and get to a certain part where they realise a lot more work will have to go into developing unforeseen pieces of code in order to get the feature to work, this will really affect our schedule, which is something that will greatly affect the overall project considering how little time we have to complete the project.

Response: Using a bit of pre-planning and SCRUM techniques, we can plan for the worst case scenario with our user story planning,

Scoping and Feature Importance:

Using Trello board to keep track of our daily progress.

Alpha Build: In our first sprint, we will spend a lot of time to create a working map generator but only 1 type of map generated, because we think the map is the basis of our entire project. In the alpha build we will also have basic troops movement and attacking, tower acquiring targets and shooting.

Beta Build: In our second sprint, our goal is to have multiple tilesets working in the map generator. Multiple maps created and layout of the game world implemented. Multiple types of troops working and spawnable. Main menu implemented and all sounds implemented

Gold Build: All memory leaks don't exist, any optimisations done to the project data structures and algorithms. Bug finding and fixing as well along with game balancing.

Third-party APIs and Middleware:

Focus on scalable APIs: Do not have to redo your existing third-party integrations when you select flexible options.




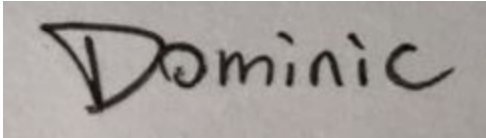
Use APIs that are well tested: With well-tested APIs from reputable sources, you can test and manage your mission-critical applications and deliver robust user experience.

Audio Middleware (Fmod): FMOD.io is a curated library of high quality sound effects for games.

Bibliography

1. Yu, D. (2016). Spelunky.

Team Signatures

	<u>6/10/2017</u>
	<u>6/10/2017</u>
	<u>6/10/2017</u>
	<u>6/10/2017</u>