Pirkanmaan Valloitus

External Documentation for Programming 3 Course

Hermanni Rytkölä & Joona Perasto

Faculty of Information Technology and Communication Sciences
Tampere University
Finland
1.12.2019

Contents

1 Introduction

This document contains an overview of all important rules and mechanics of the Pirkanmaan Valloitus strategy game. Included is also information on how the game has been designed and how it can be modified.

2 Game overview

Pirkanmaan Valloitus is a turn-based strategy game where you must expand your newfound empire and ultimately build the colossal Victory Monument building to win the game. To achieve this goal, players must build buildings and train workers, making the most of the natural resources available in their starting region. The game can be played alone but multiplayer matches of up to 4 players can also be played. In multiplayer, the first player to build the Victory Monument wins.

2.1 Resources

The game has five resources: gold, food, wood, stone and ore. Each player starts in a random area on the map with 200 gold, 200 food, 200 wood and 100 stone. The resources can be spent on buildings or workers to increase your resource production.

Gold is used to build nearly every building and worker. Gold is also spent as an upkeep cost for most buildings and workers. A player with a Marketplace building can also use their gold to buy other resources. Further information on this mechanic is available in section 3.1.

Food is used to build some buildings and train workers. It is the main limiting factor of expanding your empire; colonies require some food to build and consume a large amount of food per turn. If you do not have enough food income, you will be unable to expand. Food is generated by Farms and most tiles.

Wood is used in the construction of most buildings. It can be acquired by building lumber camps in forested tiles. Wood is also spent by the Factory building to generate a large amount of gold per turn.

Stone is a building material used in mid-late game buildings. Stone can be acquired by building any type of mine. Maintaining a steady supply of stone from the start of the game is often a good idea.

Ore is a late-game material used to build advanced buildings and workers. It can be gained by mining or bought for a premium at the Marketplace building.

2.2 Rules and instructions

The main goal of the game is simple: you must expand your empire until you are able to build the Victory Monument. Up to 4 players can play on the same map, competing against each other for limited resources. The first player to build the Victory Monument building wins. The game may continue until anyone with a Victory Monument building presses the End Game button, accessible from the Victory Monument menu. Multiple players can build the Victory Monument, but only the first player to build the Victory Monument is recorded.

The Victory Monument building is very expensive, so you will have to expand your empire to be able to afford it. Players can only build buildings within their claimed tiles. At the start of the game, every tile in a 2 tile radius around every player's city will be claimed to them. Additional tiles can be claimed in a similar fashion by building the Colony building. Your initial goal in the game should be to use the Colony building to expand your domain towards useful resources such as minerals, wood or stone.

The Colony building is moderately expensive and consumes a large amount of food per turn when built. To continue expanding your empire, you must be able to feed your populace. This can be accomplished with the Farm building, which can be built on grassland.

Citizens can be placed on tiles to increase their resource production. Basic citizens can be trained at City from the very start of the game. More efficient Educated Citizens can be trained by building the expensive University building.

All tiles support one building and one worker. Buildings can be demolished to replace the current building with another.

3 Features

The game implements many additional features that are not necessarily listed in the project grading description. This section will go through the marketplace system, 2.5D tile graphics, world generation and screen-game coordinate mapping.

3.1 Marketplace system

By building a marketplace within your empire, players can sell their resources for gold or buy resources in return for gold. Selling resources will increase the stock in the marketplace system of that specific resource. By having high stock, the marketplace will sell resources for a lower price and buy for a higher price. Vice versa, by having low stock, the marketplace will sell resources for a higher price and buy for a lower price. The marketplace prices are global and shared for every player.

3.2 2.5D tile system

3.2.1 Graphics

All of the game tiles and buildings were hand modeled in the 3D modeling program Blender. They were then rendered into a texture atlas, from which the tiles could be drawn simply by mapping game object types to their respective X and Y coordinates in the texture atlas. An overview of the graphics can be seen in figure 1 below.

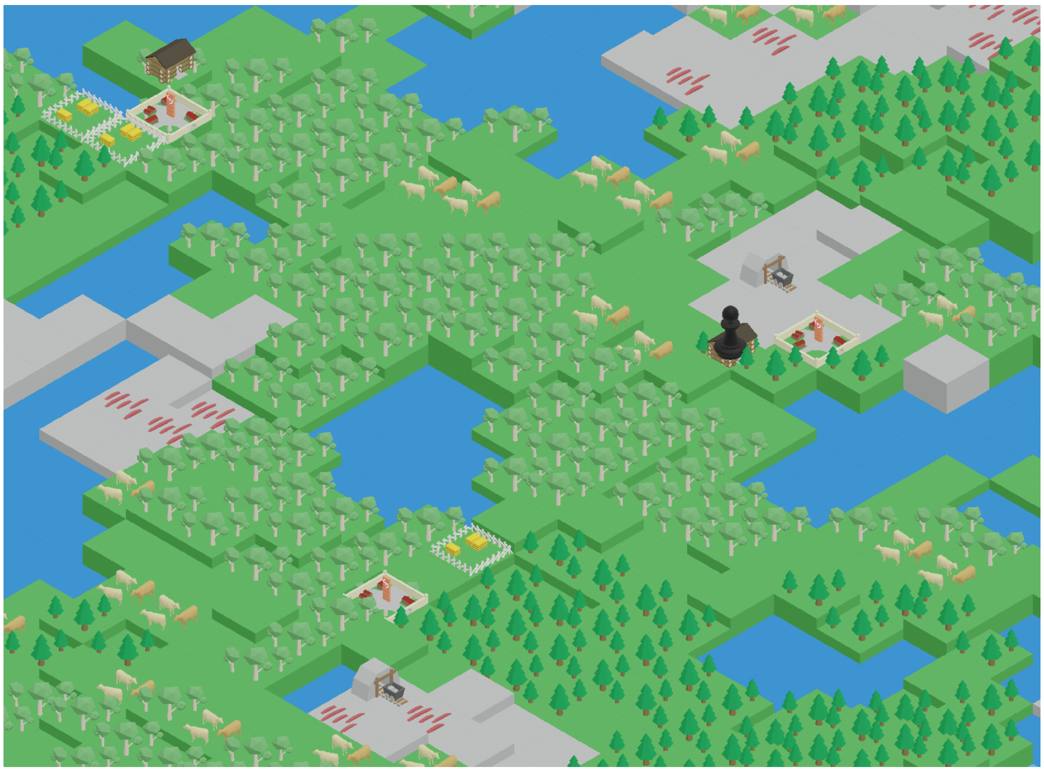

Figure 1: Game graphics

The user is able to pan and zoom the game view, which allows the user to focus on their own empire or have an overview of the entire map.

The game also features a tile highlighting system, where all the claimed tiles by current player are highlighted slightly so the user knows where they can place their buildings. The highlighting is updated in realtime when claimed tiles change, which occurs after building a new Colony.

3.2.2 Natural world generation based on random noise

The game world is generated using randomized noise using a seed given by the player. Several random values, such as height, forest level and stone level are assigned to each tile. These values are then averaged and compared to several preset constants in order to create a landscape that looks more natural than pure random noise. Rare tiles such as Ore and Diamond also have an additional percentage roll to ensure interesting distribution. The world generator was originally written in Python as a separate project and converted to C++ for the game. Because of the nature of our world generator, there was no reasonable way to inherit from the premade WorldGenerator singleton.

3.2.3 Screen-game coordinate mapping

To make this tile system possible, translation from screen coordinates to game coordinates and vice versa was required for the system to be playable. The original idea was to rotate everything 45 degrees and then squash everything to connect the top surfaces of the tiles together. However, it was just easier to move the tiles on both X and Y axes depending on their coordinates. For every X coordinate, move the tile to the right by half of the sprite's width. For every Y coordinate, move the tile down by half of the sprite's height of its top surface portion of the sprite. Tile height wasn't taken into account, but it works very well without it.

3.3 Tiles

The game has 8 different types of tiles. Each tile generates a different amount of resources and only certain buildings may be built on each tile.

- Animals
- Birch
- Diamond

- Evergreen
- Grass
- Lake
- Ore
- Stone

3.4 Buildings

The game has a total of 14 buildings, 3 of which are direct upgrades to other buildings. The buildings are listed below:

- Advanced Farm
- Advanced Lumber Camp
- Advanced Mine
- City
- Colony
- Diamond Mine
- Factory
- Farm
- Lumber Camp
- Marketplace
- Mine
- Ore Mine
- University
- Victory Monument

3.5 Workers

There are two types of workers in the game: basic Citizens and Educated Citizens. Basic Citizens can be trained from the City building, which is spawned at the start of the game for every player. Educated Citizens are far more efficient than basic Citizens in terms of resource production, but they can only be trained from the expensive University building.

3.6 Sound effects

The game uses the QSound library to play context dependent sounds. There are four sounds implemented: a sound is played when ending the turn, selecting a tile, pressing a button on the UI and when trading resources on the market.

4 Division of labour

At the beginning of the project, an initial work division was created by making a rough class diagram of the game as shown in figure 2.

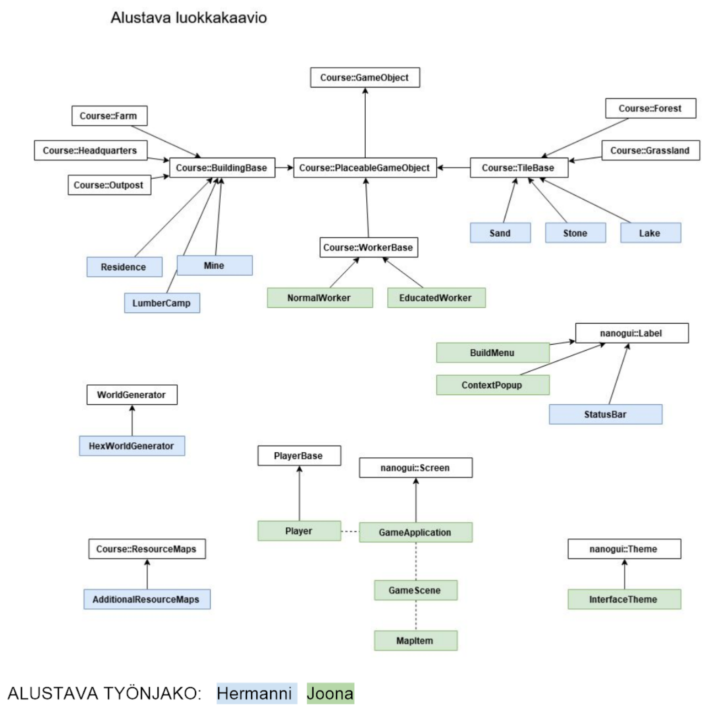

Figure 2: Rough class diagram made before starting project

We were planning to use a separate library called NanoGUI, but there were complications to get it to work. In addition to that, one of the learning objectives for the course was to get to know the Qt GUI system, so this class diagram was scrapped.

In the end, the work division was roughly split between graphics implementation and gameplay mechanics. **Joona** worked primarily on graphics implementation, GUI design, 3D modelling tiles and buildings. **Hermanni** worked primarily on gameplay, tiles, buildings, workers, and UI button implementation.

5 Grading

In this section, conditions for grading the project will be discussed.

5.1 Minimum requirements

The game project fulfills all the minimum requirements, which are listed below:

- Game compiles successfully.
- Players have turns.
- Game has dialog window at the beginning to determine player amount, world seed and map size.
- Game has multiple original Tile-classes.
- Game has multiple original Building-classes.
- Game has unit tests for ObjectManager and GameEventHandler.

5.2 Intermediate requirements

The game project fulfills all the intermediate requirements, which are listed below:

- Meets minimum requirements.
- Graphics fully implemented and no course-sided drawing is used.
- Multiple different unique Tile-classes, as listed in section 3.3.

- Multiple unique Building-classes, as listed in section 3.4.
- Two unique Worker-classes, as listed in section 3.5.
- Bonus points should compensate by 1 points.

5.3 Top-grade requirements

- Meets minimum and intermediate requirements.
- Has multiple additional features, as listed in section 5.4.
- Bonus points should compensate by 1 points.

5.4 Additional features

The game has a marketplace system, as detailed in section 3.1. The game is presented in beautiful 2.5d graphics, as detailed in section 3.2. The game has basic sound implementation for numerous features, detailed in section 3.6. Also, the game has natural noise-based world generation, which is detailed in section 3.2.2.

5.5 Bonus points for extra work

5.5.1 Graphical User Interface

The game has a very intuitive user interface. There's a side menu that contextually changes depending on what tile is selected, so you can only perform appropriate actions, which prevents user errors. The UI also gives immediate feedback on the users' actions through audio and visual feedback. There's a status bar at the top, which tells the user how many resources they have and the status of the game, including turn count and whose turn it is.

5.5.2 Graphics

Referring to section 3.2.1, the game has great handcrafted graphics modelled in 3D and rendered to a texture atlas. Extra work was also put into translating screen coordinates to game coordinates and vice versa. Tile highlighting also took quite a bit of work, since Qt doesn't support stacking graphics effects.

5.5.3 Additional tiles and buildings

The game has a large amount of variation in tiles, buildings and workers, creating a visually interesting environment and allowing tactical gameplay.

6 Tutorials

This section includes simple tutorials on how to add to or modify the game.

6.1 Adding a new sprite

This is a step-by-step guide to adding a new sprite that is bound to a tile, building or worker.

1. Pick an empty spot in the sprite sheet image for your sprite. The sprite sheet is divided into a 8x8 grid, each square being 256x256 pixels.
2. Edit the sprite sheet image in that square.
3. Finally, add a sprite mapping to the `SpriteMap` of `sprite.cpp` to its respective X and Y values.
    - Example: `{"YourSprite", spriteRect(x, y)}`

6.2 Adding a new tile

This is a step-by-step guide to adding a new tile to the game.

1. Add resource generation values for tile in `Game/Game/core/resources.h`
2. Create a new tile class in `Game/Game/core/tiles` (use existing tiles as example)
3. Add constructor in `Game/Game/core/mapgenerator.cpp`'s class constructor
    - Example: `addConstructor<Sand>("Sand");`
4. Add tile generation rules in Game/Game/core/mapgenerator.cpp's generateMap function using an existing noise function or create your own
5. Add an entry for the tile type in `Game/Game/core/objectmappings.h`'s `TILE_BUILDING_MAP`

6. Add buildings that can be built on the tile into the map (buildings must have the tile type listed in their `canBePlacedOnTile` function)

7. Add an entry for the tile type in gameeventhandler.cc's tile focus map

   - Example: `tile_focuses["Sand"] = Course::BasicResource::MONEY`

8. Set a sprite for the tile in Game/Game/graphics/sprite.cpp

   - Example: `{"Sand", spriteRect(7, 0)}`

6.3 Adding a new building

This is a step-by-step guide to adding a new, fully functional building to the game.

1. Add resource generation values for tile in `Game/Game/core/resources.h`

2. Create a new building class in `Game/Game/core/buildings` (use existing buildings as example)

3. In `Game/Game/core/objectmappings.h`, add your building to every tile map where you want it to be possible to build (buildings must also have the tile type listed in their `canBePlacedOnTile` function)

4. Add the building's build cost in the BUILD_COSTS map

   - Example: `build_costs["Port"] = ConstResources::PRT_BUILDCOST;`

5. Add the building's constructor in the BUILDING_CTOR_MAP

   - Example: `"Port", makeConstructor<Port>(),`

6. set a sprite for the tile in Game/Game/graphics/sprite.cpp

   - Example: `{"Port", spriteRect(5, 2)}`

7 Troubleshooting

The game was primarily designed on Windows and may have issues running on Linux. Here are fixes to common errors if you cannot get the game to run on Linux:

- **A million missing header files:** remove the UnitTests subproject from the main project.

- **Performance issues, low framerate:** the game could use some optimization, but on our PCs running Windows, it runs fine. On Linux remote desktop it is still playable but quite choppy. Unfortunately not much can be done about this.
- **`PulseAudioService: pa_context_connect() failed:`** system does not support audio devices, this is outputted when trying to play a sound.