Colonizing Pirkanmaa Game Documentation

By: Team Flatmate

Team members:

Tran Hoanh Le – 281721 – hoanh.le@tuni.fi

Ben Kavanagh – 281654 – benjamin.kavanagh@tuni.fi

1. Introduction

Colonizing-Pirkanmaa game is a course project given in TIE-02408. The game is based on an existing board game which the course staff provided base classes for building, tile and worker as well as other core functionalities. The task of our team is to implement the graphical user interface, define details of game play and implement them.

Our team has implemented the minimum, intermediate requirements and top-grade requirements, as well as two additional features and some extra work which qualify for bonus points. Section 2 describes the software architecture and details the classes implemented by our team. Section 3 describes the implementation of extra features in our game and extra works. Section 4 provides information about the workload division. Sections 5 details the gameplay implement by our team. Section 6 concludes the documentations.

2. Software Architecture

The game is implemented by both our team and course staffs. Details of the code implemented by course staff can be found here (or in *Documentation/Course_Doxy-documentation*).

All the implementations by the course side are in the namespace **Course**. Our team implementation resides in two main namespaces **Student** and **Ui**. Doxygen files of our implementation can be found here (or in Documentation/Student_Doxygen-documentation).

THE PROJECT STRUCTURE OF CLASSES IMPLEMENTED BY COURSE STAFF

- **Building**
  - **BuildingBase**: represents base-class for different buildings in the game
  - **Farm**: represent farm production buildings in the game, which create resources for the player.
  - **Outpost**: represent outpost building, which the player can buy to claim ownership of neighborhood tiles.

- **Core**
  - **PlayerBase**: represents base class for Player class.
  - **PlaceableGameObject**: represent Gameobject that can be placed on Tile
  - **Gameobject**: a base-class that contains general information on different objects in the game
  - **BasicResource**: implement resource operations and structure in the game
- **Exception**
  - Implementation of the following exception in the game: illegalaction, invalidpointer, keyerror, noteenoughspace, ownerconflict
- **Tile**
  - **Forest**: represent Forest Tile in the gameworld.
  - **Grassland**: represent Grassland Tile in the gameworld.
  - **TileBase** represent base-class for all different tile-classes in the game.
- **Workers**
  - **WorkerBase**: represent abstract base-class for all worker-objects.

THE PROJECT STRUCTURE OF CLASSES IMPLEMENTED BY US

- **Building**
  - **HousingBase** (and other derived classes from **HousingBase** – **ApartmentBlock, LargeHouse, SmallHouse, Skyscraper**): represent housing buildings in the game, which add different number of workers to the player.
  - **Mine** and **Sawmill**: represent production buildings in the game, which create resources for the player.
- **Core**
  - **GameEventHandler**: handles events happening in the Mapwindow such as create players in the beginning, assign workers from the player to a tile, check if a player has won.
  - **Gamescene**: provides the interface to display the game map and allows the user to interact with the tiles there.
  - **ObjectManager**: Keeps track of the game objects positioned to the board including the tiles, buildings and workers.
  - **Player**: represents a player in the game, which is used to store and access NewBasicWorker object and Tile object and keep track the Resources and Workers the player has in the game.
- **Exception**
  - **NoOwner**: Is derived from the base exception class for instances when an object has no owner.
- **Graphics**
  - **AssignDialog**: implement a dialog to take the number of workers the player wants to assign to the selected Tile.
  - **HighScoreDialog**: implement a dialog to show the top 5 scores of all time. Score is the number of turns the player takes to get 5000 resources total and 50 workers in total. The lowest number of turns counts as the best score.

    - **Mapitem**: An item that can be positioned on the map such as a building or tile.
    - **Mapwindow**: implement the main GUI of the game, where the players do actions on. This class gets the handler.
    - **RulesDialog**: implement a dialog to show the players instruction on how to play the game.
    - **SetPlayerDialog**: implement a dialog to take the number of players in the game.
    - **StartDialog**: implement a dialog to show in the beginning of the game, where the players can go to RuleDialog to see the instructions or start the game and go to Mapwindow.
    - **UnAssignDialog**: implement a dialog to take the number of workers the player wants to unassign from the selected Tile.
    - **WinDialog**: implement a dialog to display winning message after a player has won, asking if the players want to play again or exit the game.
- **Tile**
    - **Rock**: represent Rock Tile in the gameworld.
    - **Sand**: represent Sand Tile in the gameworld.
    - **Water**: represent Water Tile in the gameworld.
- **Workers**
    - **NewBasicWorker**: represents new basic worker in the game, which the players can only acquire by buying housing-type building.
    - **Farmer**, **Logger** and **Miner**: represent farmer, logger and miner worker in the game, each has different efficiency and production-focus.

3. Extra features and works

Along with minimum and intermediate requirements, we also implement the following extra features and works:

- Highscore system: Display the top 5 scores of all time in the game. The score database is saved in scoreDB.txt. Every time after a player wins, the score is recorded into the file. When the player wants to see the highscore, it will show the best scores (maximum 5) on the file scoreDB.txt. This is implemented in Ui::Mapwindow. Note, in order for the highscore dialog to work it requires the build settings to uncheck build in shadow mode. This enables the program to find the score database.
- New game mechanics: Instead of buying more workers and food and money, we implemented so that if the player can only have more workers by building housing-base buildings. This is implemented in Student::HousingBase and derived classes from that. All workers classes are implemented so that they have no cost to buy.
- Complete Game GUI: the game has complete GUI which consists of startdialog to start the game, rulesdialog to show how to play, setplayerdialog to set the number of players in the game, mapwindow to display the map where players play, assigndialog and unassigndialog to assign and unassign workers and windialog

when a player won and ask if the player wants to play again. Below is the navigation diagram in our game.

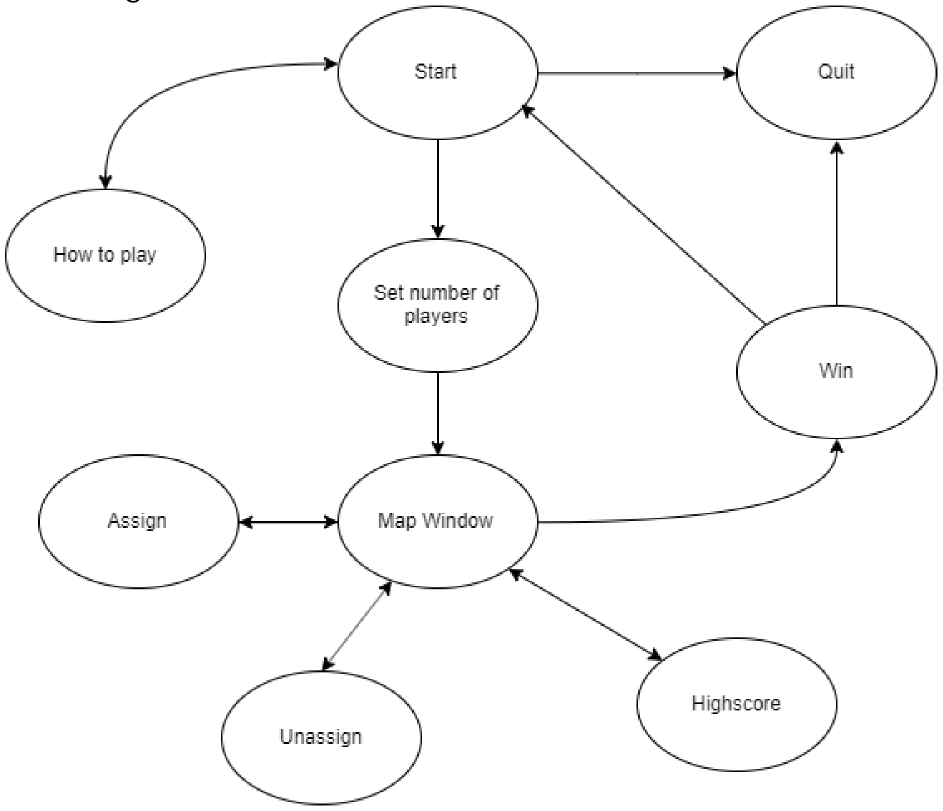


Figure 1 Navigation diagram of our game

- We implemented in total of 6 new buildings (apartment, smallhouse, largehouse, skyscraper), 4 new workers (newbasicworker, logger, miner, farmer) and 3 new tiles (rock, sand, water). Cost and production of those can be found here.
- Back story and depiction of the game world can be found here. (Or in Documentation/back_story.txt).

4. Workload division

In the beginning, we had a meeting to discuss the details of the game play as well as the game implementation plan and explored the codes implemented by the course staff. Here is our workload division plan and we followed the plan closely:

- **Ben**:
  - Mapwindow interface design. (design where the blocks are, graphics of buildings and worker buttons and tiles)
  - Gameobjectmanager class.
  - Tile-classes.
  - Gamescene and mapitem classes

- Unittest of gameobjectmanager and gameeventhanlder

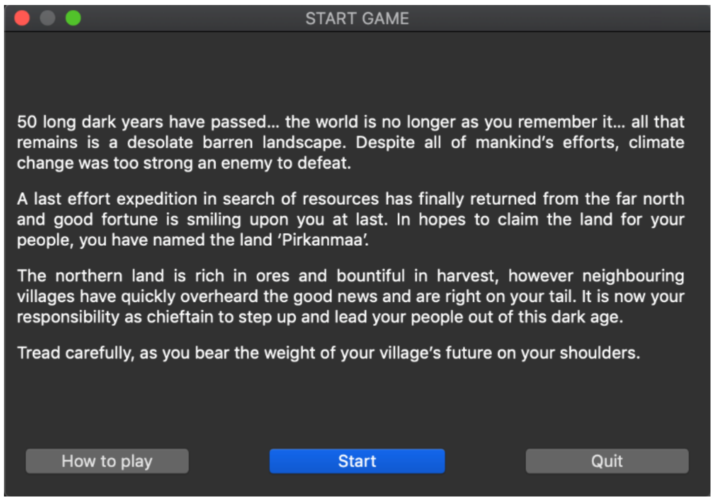
- **Hoanh**
  - All necessary dialogs design and implementation.
  - Players can restart the game
  - Gameeventhandler class.
  - Housingbase class.
  - Player class.
  - Worker-classes
  - Saving score and show the top 5 scores on highscoredialog.
  - Assign and unassign button of mapwindow behaviours.
  - Update resources and workers information on mapwindow.

- **Both**
  - Design the logic and flow of the game
  - Design the game story and play
  - Fixing bugs and crashes
  - Write the software documentation

5. Game Manual

Game-start:

Users are greeted with the back story and can click start to proceed. In the next window the user can choose the number of players in the game (from 2 to 4 players).



50 long dark years have passed… the world is no longer as you remember it… all that remains is a desolate barren landscape. Despite all of mankind's efforts, climate change was too strong an enemy to defeat.

A last effort expedition in search of resources has finally returned from the far north and good fortune is smiling upon you at last. In hopes to claim the land for your people, you have named the land 'Pirkanmaa'.

The northern land is rich in ores and bountiful in harvest, however neighbouring villages have quickly overheard the good news and are right on your tail. It is now your responsibility as chieftain to step up and lead your people out of this dark age.

Tread carefully, as you bear the weight of your village's future on your shoulders.

Figure 2: The start game dialog

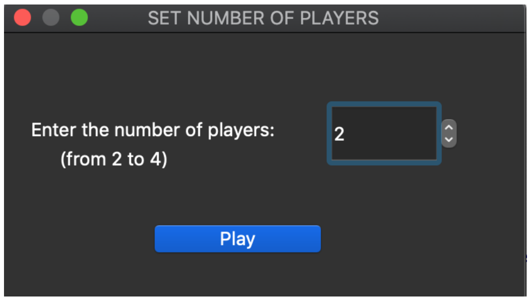


Figure 3 The dialog to select the number of players

The game is a turn-based game. The players can decide between them the order of which they play. For example, play rock-paper-scissors. In the beginning, each player will have 500 resources each type (food, money, ore, wood, stone). This money can be used to buy and build buildings on the map, each of which have a respective build-cost and production benefits.

Gameplay:

Each turn, players can do those following actions:

Build buildings:

There are three main types of buildings in the game.

- Sawmill, farm, mine. They help the users to increase the production of each land (tile) the player has
- SmallHouse, LargeHouse, ApartmentBlock and Skyscraper. Build these buildings increase the number of people (workers) the player has.
- Outpost. Gives the players ownership of neighbourhood tiles.

How to build

The player must select a tile that has no-owner or which he/she owns the tile/land. After that, the player can pick the building type they wish to build and then press build button to build. (Remember: player can only build if he/she has enough resources). Note it is also good to consider where you are building a building as each tile type has a base production rate. Note buildings cannot be built on water.

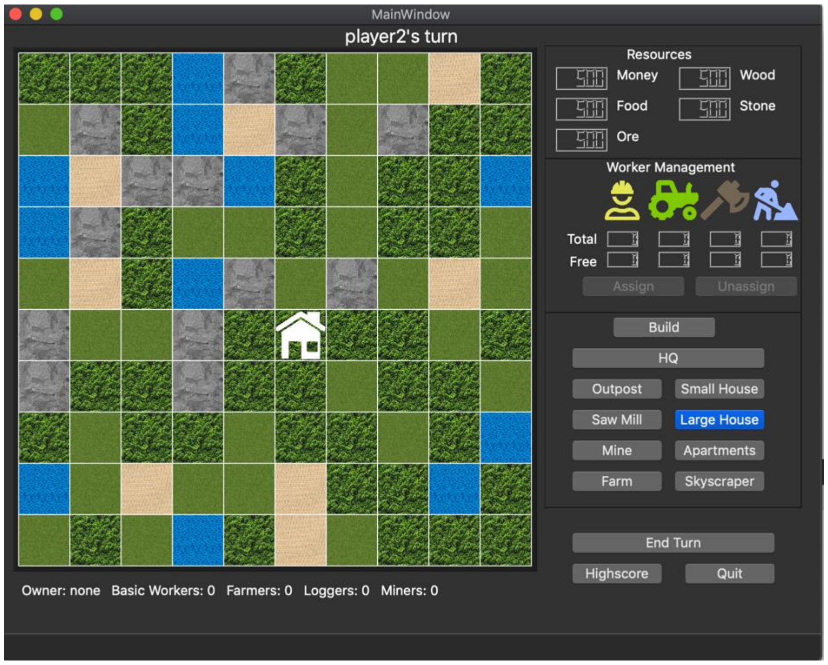


Figure 4 Here a tile and building type has been selected so the build button is available

Building build costs:

SmallHouse: Money 50, Food 100, Wood 25
LargeHouse: Money 100, Food 150, Wood 35, Stone 500
ApartmentBlock: Money 250, Food 200, Wood 100, Stone 100
Skyscraper: Money 1000, Food 500, Wood 500, Stone 500
Farm: Money 50, Food 100, Wood 25
Mine: Money 200, Food 100, Wood 100, Ore 50
SawMill: Money 100, Food 100, Wood 100, Ore 50
Outpost: Money 150, Food 200, Wood 200, Stone 25
HeadQuarters: Money 750, Food 1000, Wood 500, Stone 250

Building production rates:

Farm: Money 1, Food 20
Mine: Money 20, Stone 50, Food 5, Ore 35
SawMill: Money 20, Wood 50, Food 5
Outpost: Money -5, Food -2
HeadQuarters: Money 10, Food 2

Tile base production:

Grassland: Money 2, Food 5, Wood 1, Stone 1, Ore 0
Forest: Money 1, Food 3, Wood 5, Stone 1, Ore 0
Sand: Money 1, Food 1, Wood 0, Stone 1, Ore 2
Rock: Money 3, Food 0, Wood 0, Stone 5, Ore 2

Housing Capacity:

SmallHouse: 4 workers (1 of each type)
LargeHouse: 8 workers (2 of each type)
ApartmentBlock: 12 workers (3 of each type)
Skyscraper: 24 workers (6 of each type)

Assign workers:

The player can assign workers to a tile to increase the efficiency of the tile-production. The total number of each worker type and the number of available workers can be seen below the worker buttons.

How to assign:

The player can click on tile to see information about that tile. If the player owns the tile, he/she can assign the workers to tile by choosing the type of workers he/she want to assign and click assign button. After that, a dialog will open and the player can type in the number of workers to assign.

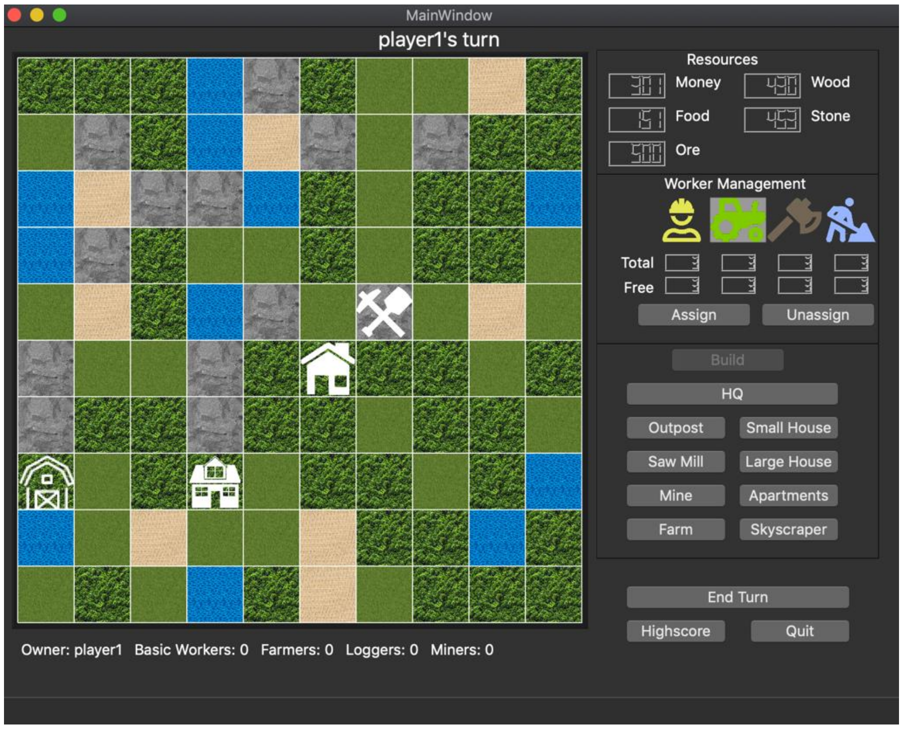
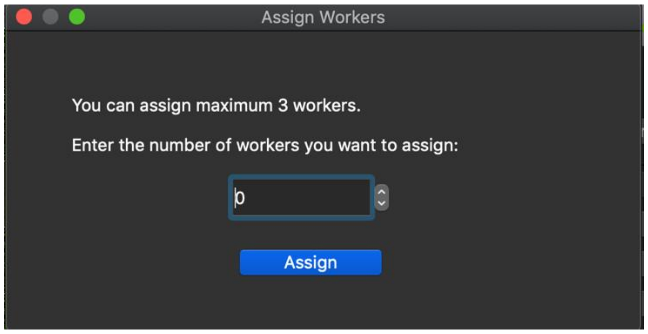


Figure 5 A tile and worker has been selected so the assign button is available



Assign Workers

You can assign maximum 3 workers.

Enter the number of workers you want to assign:

0

Assign

Figure 6 Assign worker dialog

Unassign workers:

The player can unassign workers from a tile to get them back to assign to another tile.

How to unassign:

The player can click on tile to see information about that tile. If the player owns the tile, he/she can unassign the workers from the tile by choosing the type of workers he/she want to unassign and click unassign button. After that, a dialog will open and the player can type in the number of workers to unassign.

Worker Production Multipliers:

Basic Worker: Money 0.25, Food 1, Wood 0.75, Stone 0.50, Ore 0.50

Farmer: Money 0.50, Food 1.50, Wood 0.50, Stone 0.25, Ore 0.25

Miner: Money 0.50, Food 0.25, Wood 0.25, Stone 1.00, Ore 1.00

Logger: Money 0.50, Food 0.25, Wood 1.50, Stone 0.25, Ore 0.50

The player can end their turn by clicking the end turn button.

The aim of the game:

The game ends when a player can get 5000 resources in total and 50 people in total. His/her village becomes the largest in Pirkanmaa and they claim the ownership of the whole magical land!

6. Conclusion

Currently, there are no bugs at the time of writing the documentation. Longer time of game testing may reveal bugs if there are any.

However, we find that our game can be further developed in the following points.

- First, currently, the player can only see the details related to resources such as how much a building cost, how much a building/tile produce or efficiency of the workers in RulesDialog, not in the Mapwindow. In the future, we can implement the game so that it can show the information as well in the mapwindow.
- Second, the size 10*10 might be small for 4 players and it might be hard to win the game. In the future, we can have another dialog to configure the game so that the players can also change the size of the map and the goal of the game (instead of 5000 resources in total and 50 workers in total) as well as the name of the players.
- Third, to increase the interaction between players, we can implement commerce system in our game so that the players can exchange resources to each other. (for example: exchanging 100 foods for 50 stones).

As the game is currently in version 1.0, there is still lots of room for improvement. However, it provides a good foundation to build upon.