Island Sinking Game Documentation
Team DK

Dat Thanh Tran – 250820 – dat.tranthanh@tut.fi
Duy Khoa Nguyen – 272580 – nguyen29@student.tut.fi

Version 0.2.0

1. Preface

Island Sinking Game is a course project given in TIE-02408. The aim of the project is to practice design-by-contract principle in object oriented programming paradigm. The game is based on an existing board game and the course staff provides necessary interface and parts of the functionality. The task of each team is to implement to user interface, define details of the game play and implement them.

Our team had completed the minimum and basic requirements, as well as some five extra features. Section 2 describes the whole software architecture and details the important classes implemented by our team. Section 3 describes the implementation of extra features in our game. Section 4 provides information about the planned and actual workload division. Section 5 details the game play implemented by our team and Section 6 concludes this documentation with know bugs or missing features.

2. Software Architecture

As mentioned previously, the game is a joint effort between the course staffs and our team, in which common interface and functionalities are designed by the course staffs. Details interface provided by the course staff can be found via the following link: http://www.cs.tut.fi/~otekn/2018/doxygen/

All the logics given by the course side lie in the namespace **Common**. Our team implementation resides in two namespace **Student** and **UI**.

In short, **Common** namespace implements the logics of game pieces as well as a game engine:

- **Hex**: this class implements the functionality of a hex tile
- **Pawn**: this class implements the functionality of pawns of player, .e.g., coordinate getter/setter, set ID...
- **Actor**: this class acts as the base class that defines common functionality of actors (Vortex, Shark, Seamunster, Kraken), e.g., move, do action…
- **Transport**: this class acts as the base class that defines the functionality of transports (Dolphim, Boat), e.g., add pawn, remove pawn, move, check capacity…
- **GameEngine**: this class implements the basic functionalities of game play such as checking the validity of a particular movement, adding/removing game pieces…

Student namespace holds the implementation of three classes that are the backbone of the game:

- **GameState**: this class keeps track of the state of the game with interface designed by the course staff (*IGameState*)
- **GameBoard**: this class acts as the datastructure of the game elements, also with interface designed by the course staff (*IGameBoard*)
- **Player**: this class implements the datastructure for the players, with interface designed by the course staff (*IPlayer*)

UI namespace holds the implementation of all UI elements, including:

- **ConfigurationWindow**: this class implements a dialog to take configuration inputs from users (names of players and number of pawns per player)
- **GameOver**: this class implements a dialog to display game over message, asking whether user wants to replay or exit the game.
- **GraphicHex**: this class implements the visualization of **Common::Hex**
- **GraphicPawn**: this class implements the visualization of **Common::Pawn**
- **GraphicTransport**: this class implements the visualization of **Common::Transport**
- **GraphicActor**: this class implements the visualization of **Common::Actor**
- **ControlBoard**: this class implements the display of game state, scores, top10, current player's turn and the wheel
- **HexBoard**: this class implements the visualization of the **Student::GameBoard**. While keeping track of the UI elements of game pieces (pawn, hex, transport, actor), this class also handles the mouse events performed on those pieces. This class can be considered as the conductor of the game.
- **MainWindow**: this class holds the **HexBoard** and **ControlBoard** into a single window, performing necessary initialization of the game.

3. Extra Features

Along with basic requirements, we also implement the following 5 extra features:

- *Zoomable game board:* The game board can be zoomed by using wheel of the mouse, this allows flexible adjustment of the view of the game board. Zoomable functionality is implemented in **UI::HexBoard** by handling wheelEvent()
- *Scrollable game board*: both vertical and horizontal scrolling are support when the zoomed game board is larger than the actual size. This functionality is also implemented in **UI::HexBoard** by providing scrollbar objects to the QGraphicsScene object.
- *Visualization of wheel*: The wheel is visualized by using our drawn graphic (PNG file). This is handled by the UI element **UI::Wheel**. Since the wheel displays two outputs, we have **UI::Wheel** that implements the functionality of both the inner and outter part of the wheel
- *Animation of wheel*: When the wheel is clicked in appropriate stage (SPINNING), the visualized wheel is spinned and displays the correct output given by the game engine. Animation is also handled in **UI::Wheel**.
- *Top10 list*. Displays the top 10 players that have played the game. In every game, when the game is over or quit, the current points of all players are recorded and checked if they can be added to top10. Top10 is handled in **UI::ControlBoard**.

4. Workload Division

At the beginning, we divided the workload so that Dat implements the three main classes GameBoard, GameState and Players and both will explore the UI elements of the game. However, since the implementation of GameBoard, GameState and Players is easy, Dat moved onto UI elements quickly. The imbalance in the coding skills between the two led us to agree that Dat will implement the draft UI while Khoa will handles the unittest, CI configuration and game testing.

The actual workload done is as follows:

- The classes inherited from IGameBoard, IgameState and IPlayer. (Dat & Khoa)
- Unittest (Khoa)
- CI configuration (Khoa)
- ConfigurationWindow (Dat & Khoa)
- UI namespace (Dat)
- Extra features (Dat)
- Game Testing (Dat & Khoa)

5. Game Manual

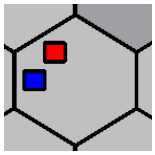
The game is turn-based and playable by two or more players. All movement are done by dragging the corresponding elements. A turn consists of three stages:

- The 1st stage is called MOVEMENT. The player moves their pawns: the player has three moves in play. These can be used to move one or more pawns.
    - The player cannot move through/to a full tile (tile that has 3 pawns).
    - When a pawn moves from an island tile, the number of move is calculated as the number of tiles that the pawn travels. However, when a pawn moves from a water tile, then it can only move to the next tile and this costs 3 moves.
    - When a pawn is moved to a transport on the same tile, this cost no move. Similarly, the player can also move pawn out of the transport on the same tile without costing any move.
    - When a pawn is dragged onto the body of a transport, it is considered as adding the pawn on the transport. If the transport is not full and the movement is valid, then this is executed, otherwise, the pawn is moved back to the old position.
    - At any moment, the player has the freedom to continue moving the pawn using the transport (if pawn is on transport) or moving out of transport and swimming.
    - If the pawn is dragged onto the boat and moved by the boat, the boat can hold maximum 3 pawns and the player with most pawns on the boat can move the boat. The cost of the movement is similar to moving the pawns on island, i.e., the number of move is counted as the number of tiles that the boat travels. The only exception is that the boat cannot move to a tile that already has another transport
    - If the pawn is dragged onto the dolphin and moved by the dolphin, the dolphin can hold only 1 pawn at a time. The number of moves is calculated similar to the boat. The only exception is that the dolphin cannot move to a tile that already has another transport
- The 2nd stage is called SINKING. The player chooses an island tile that will sink. First the beach sinks (yellow color tile), then the forest (green color tile) and finally the mountains (grey color tile). Pawns on the island tile will drop into the sea. If the island has completely sunk, this stage is skipped. When a tile is sunk, an actor or a transport will appear on that tile. In case of vortex, the action is performed immediately during the second stage (SINKING)
    - vortex: destroys everything on the neighboring sea tile (blue color tile) as well as destroys everything on the current tile, even the vortex itself.

- The 3rd stage is called SPINNING. The player spins a wheel by clicking on the wheel. The wheel tells which animal moves and how much. The options are 1, 2, 3 and D. The number tells the amount of tiles the animal can be moved over. D means the animal dives and can be placed into any free sea tile.
    - Dolphin: the current player can move any dolphin on the scene with the number of movement dictated by the wheel output.
    - Shark: eats pawns and dolphin. The pawns and dolphin on the tile are removed from the game. The exception is that shark cannot move to the tile that either has boat or seamunster.
    - Kraken: destroys the boats. Pawns on boat are dropped into the sea. The exception is that kraken cannot move to a tile that has seamunster.
    - Seamunster: destroys boat and pawns. The exception is that seamunster cannot move to a tile that has shark.

The visualization of those game graphics are displayed below

Color coded Pawns

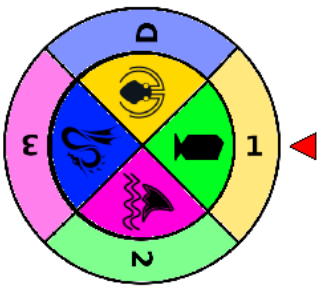


Boat

Kraken

Dolphin

Game Wheel



Shark

Seamunster

6. Bugs

We have not confronted any bugs at the time of writing. However, longer time of testing can definitely reveal bugs if there are any.