Assignment 03

Group 02

Exercise 1 - Design patterns

1. This week, we have implemented two new design patterns.
    a. **Strategy pattern:** the strategy pattern allows selecting an algorithm's behaviour at runtime.[1] We implemented this pattern to handle collisions depending on the type of units involved. Previously, collision handling was done through separate classes. The strategy pattern makes handling collisions more consistent.
    b. **Observer pattern:** the observer pattern makes it possible to publish events without having knowledge about the subscribers of those events. We implemented this pattern to handle data synchronization between the ScoreScene and the ScoreCounters (as part of the Cursors). This makes it easy to create new classes that depend on the current score of some Cursor.
2. **Class diagrams**
    a. **Strategy pattern:** In the collisions package, we added a CollisionInterface interface. This interface contains one method to be implemented, namely **boolean** intersect(Unit unit);. Every unit implements this method and specifies what should happen when some unit intersects with it. See docs/sprint3/diagrams/collisionInterface.png for an overview of how the system fits together. See docs/sprint3/diagrams/collisionInterface-full.png for a more complete overview containing all the methods and attributes of the classes involved.
    b. **Observer pattern:** The ScoreScene class now implements the java.util.Observer interface. The ScoreCounter class now extends the java.util.Observable interface. The Cursors contain a ScoreCounter which data is then distributed over the ScoreScenes. See docs/sprint3/diagrams/observer.png for an UML diagram of the classes involved.
3. **Sequence diagrams**
    a. **Strategy pattern:** There are three classes that implement the CollisionInterface interface. Because of this, we have created three sequence diagrams.
        i. docs/sprint3/diagrams/qix-intersect.png shows the interaction between Qix and Unit, Logger.
        ii. docs/sprint3/diagrams/unit-intersect.png shows the interaction between Stix and Qix, Logger.
        iii. docs/sprint3/diagrams/stix-intersect.png shows the interaction between Qix and Unit, Logger.
    b. **Observer pattern:** See docs/sprint3/diagrams/scorescene-update.png for a sequence diagram showing the calls when ScoreScene.update() is called.

1: https://en.wikipedia.org/wiki/Strategy_pattern

Exercise 2 - Your wish is my command

1. **Multiplayer gameplay requirements**

Must haves:

- Players play agains each other. Lines of the players can’t cross .
- Each player has it own scores.
- Each player has it own buttons.

Should haves:

- If a player is on a innerborder or area the player will die. This is possible because player 2 can create an area next to player 1.
- They can kill the other by capturing them with a new made area.

Could haves:

- when a player kills the other player a sound will play

Explanation:

Multiplayer is more players thus we have two players playing against each other.
So both the players need to see their scores.
To play with more players on one computer both players need to have different buttons to play on.
To make the game more competitive players can kill each other.
To make the game more exciting we added a sound.

2. **UML diagram.** See docs/Sprint #3/diagrams/multiplayer-diagram.png

Exercise 3 - 20-Time

1. **Requirements own game improvement**

Must Haves:

- The game must track how many times the player has died in the current game.
- A variable which describes how many lives a player starts with should be defined.
- If the amount of times a player died is equal to the amount of lives the player starts with the game is over.
- The game must have multiple life trackers if there are multiple players

1: https://en.wikipedia.org/wiki/Strategy_pattern

- The player must respawn where the stix began drawing or where the player died because of a collision with sparx.
- The sparx should always respawn on the opposite side of the board

Should haves:

- There should be a visual indicator which shows how many lives the player has left.
- The amount of lives the player starts with should be variable with an argument on startup.

Could haves:

- The game could have a different sounds for when a the game is over or the player only lost a life.
- The game could give the player bonus score depending on the amount of lives the player has left after completing the game

2. UML diagram

This UML has already been prepared for multiplayer because of a list of cursors so the game can have multiple cursors which can be controlled by multiple players. The following diagrams show the interaction between the three classes which have to do with the multi life system. The gamecontroller gets to know from the collisionHandler if a cursor has collided, if that is the case it tells that cursor it has died which in turn updates the scorecounter where the amount of lives of a cursor is being tracked.

1: https://en.wikipedia.org/wiki/Strategy_pattern