Assignment 3

Design Pattern

and

Software Architecture

SEM Group 45

Table of contents

Design Patterns

Design pattern 1………3

Design pattern 2………5

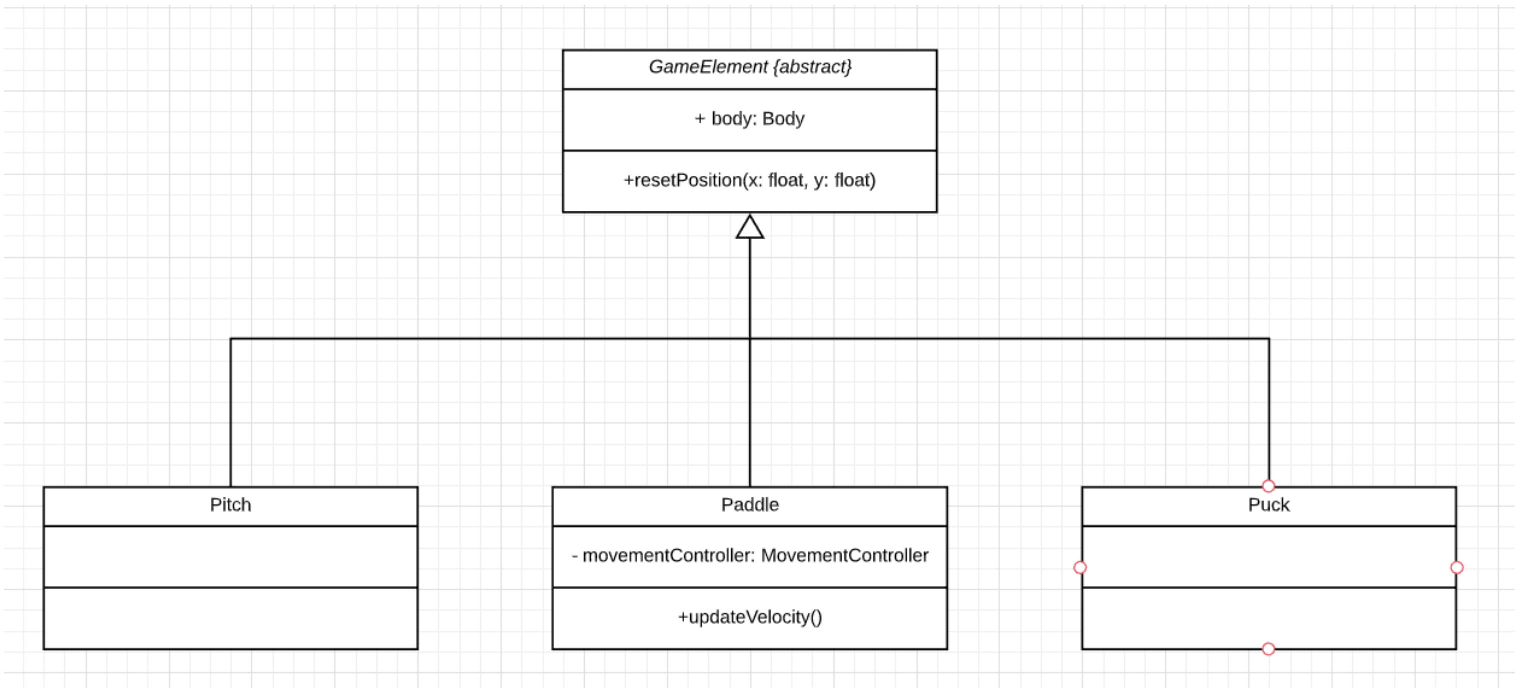
Software Architecture

Game architecture………………………………………………………………………………………………………6

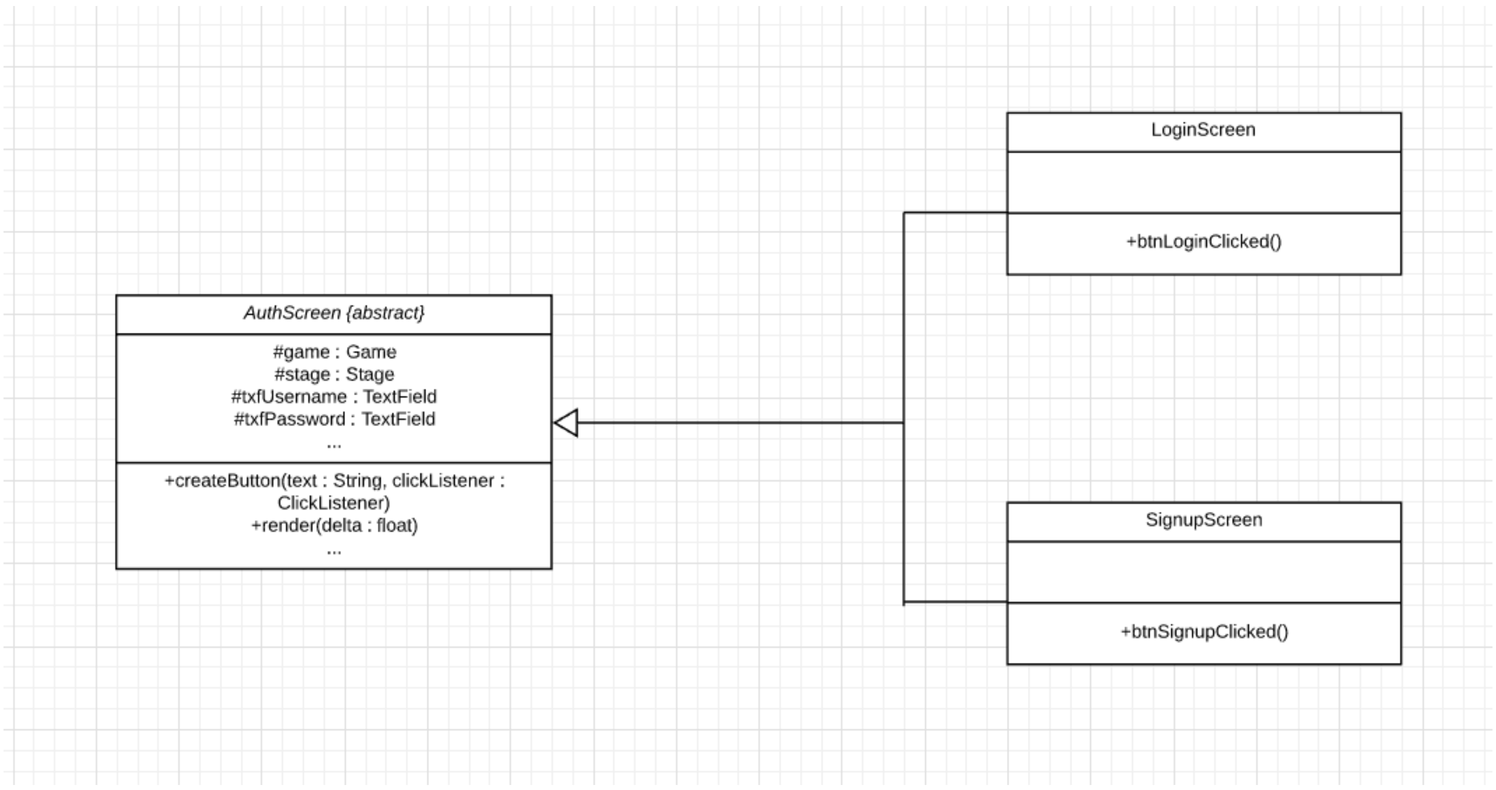
Design pattern 1

Our first design pattern is the template design pattern, which is a creational design pattern. We integrated this design pattern in our game elements and for our screens.

We implemented the template design pattern for the game elements paddle, puck and pitch because at their core they are very similar, namely in the fact that they all have a body with which something should happen. For the pitch the body will be the place where the walls are created, for the paddle the body should be controlled by the player and the puck has a body to be able to interact with the paddle. The ‘GameElement’ class contains this body attribute from which the other subclasses extend.


GameElement {abstract}
+ body: Body
+resetPosition(x: float, y: float)
Pitch
Paddle
- movementController: MovementController
+updateVelocity()
Puck

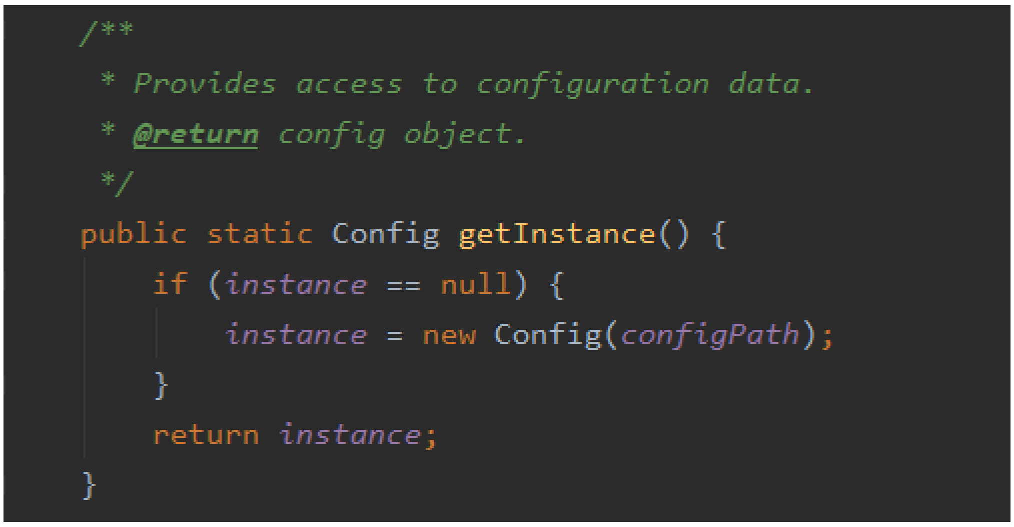
The template design pattern is also used for the authentication screens since their behaviour is extremely similar (type in a username and password and clicking a button). The login a signup screen both inherit from the authentication screen, which has two text fields for username and password and a button. The login screen has a login button while the signup screen has a signup button.



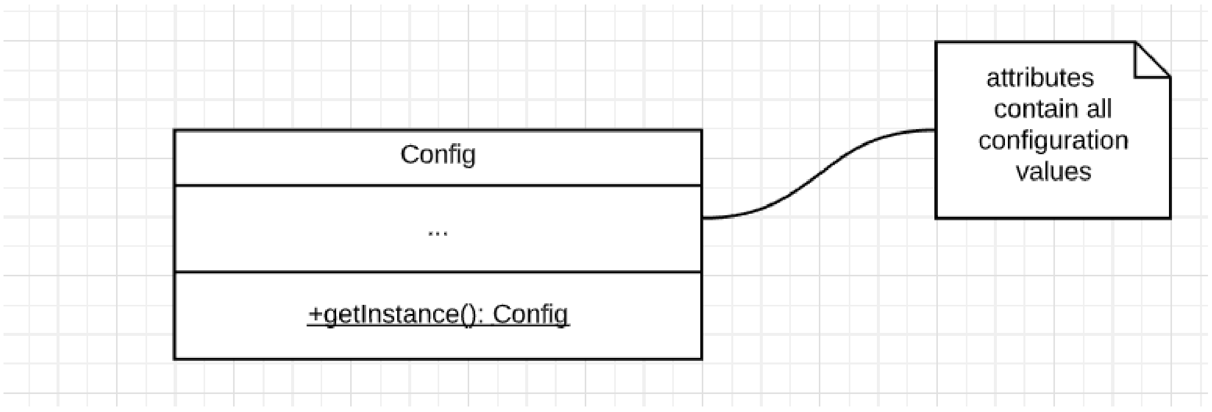
AuthScreen {abstract}
#game : Game
#stage : Stage
#txfUsername : TextField
#txfPassword : TextField
...
+createButton(text : String, clickListener : ClickListener)
+render(delta : float)
...
LoginScreen
+btnLoginClicked()
SignupScreen
+btnSignupClicked()

Design pattern 2

The second design pattern we implemented is the singleton pattern. We use the design pattern for the configuration file. Since there isn't any need for one configuration class once the program is loaded we decided to make a singleton out of the class. Furthermore, it would be a problem if there are multiple configuration classes with different values, since then there would be uncertainty as to which file is the correct one, which could lead into problems with playing the game. Basically, the singleton will ensure that there is one configuration class present which acts as global point of access for the configuration of the game.



```
/**
 * Provides access to configuration data.
 * @return config object.
 */
public static Config getInstance() {
    if (instance == null) {
        instance = new Config(configPath);
    }
    return instance;
}
```



Game architecture

The game architecture of our game is not extremely complicated since there are not that many modules that work together to make the program work. The project is pretty confined and overall does not require many resources. Nevertheless, there are still some parts on a high level that interact with each other that make up the program.

The database contains all the login information of the players. It also holds the scores of the registered players. Furthermore, whenever a new player signs up, their credentials are stored in this database. Also, at the post game screen the top 5 scores are retrieved from the database. So, whenever a player is using the login screen, the signup screen or ends up at the post game screen, the system is interacting with the database.

The physics of the game are provided by an external library, namely the libGDX (GDX library). All actions that involve physics are handled by this library. This means that the backend of our game depends on this external library to work. The library also provides other elements to both the backend and GUI of our game, for example 'screen' classes from which we extend all our screens like login screen, signup screen and game screen. The library also provides some classes which make it easy to create a world and game.

We choose to use a library instead of making all the elements just mentioned ourselves because building these from scratch takes a lot of time, which we simply do not have. In addition, the elements we would create ourselves would probably end up similar to the one provided by the library, only with much less features since these classes can become very complicated.

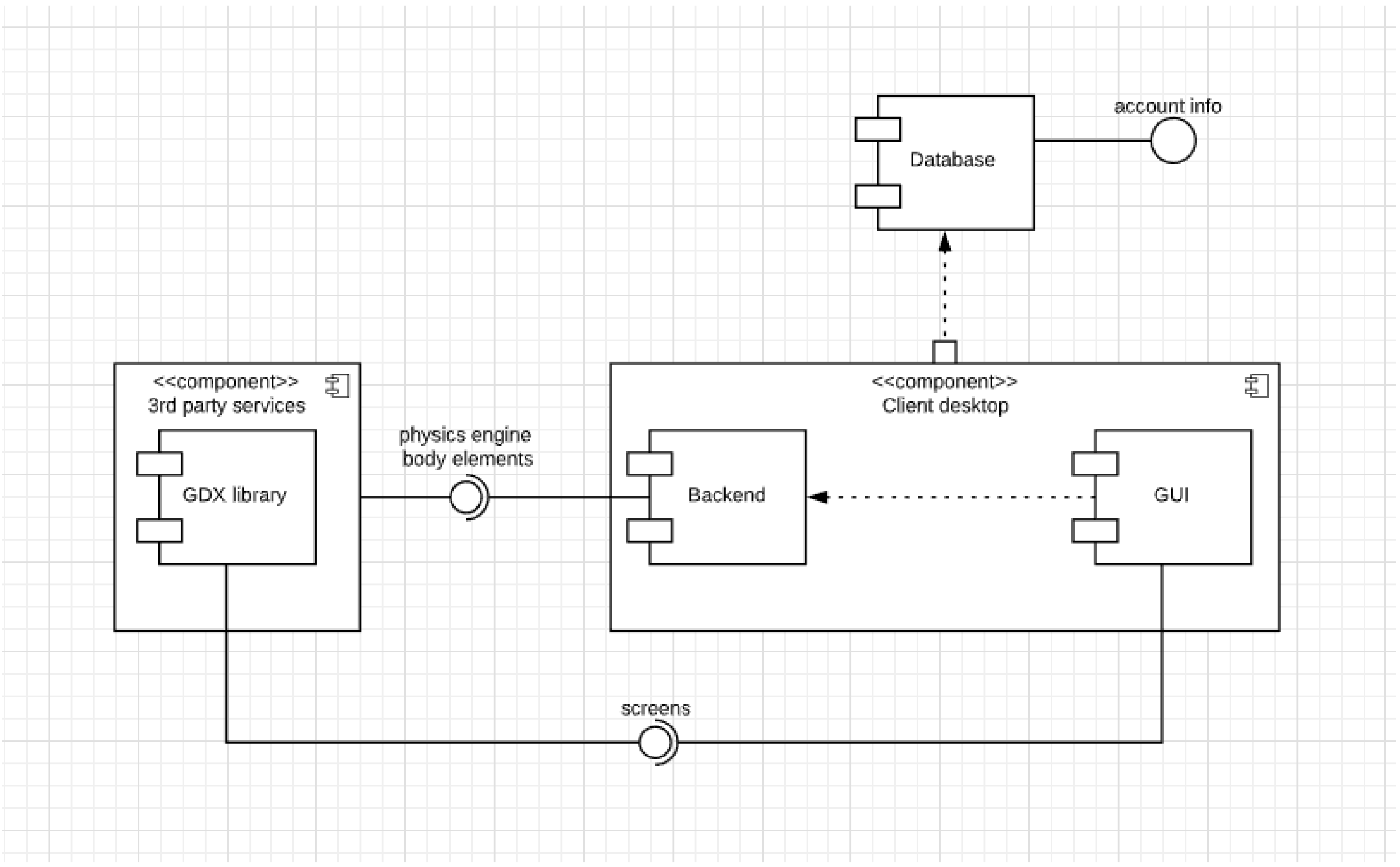
As already mentioned, we also have a GUI which also makes use of the GDX library. In particular, the library provides a 'screens' class which we use to present our program. In these screen classes we create buttons, text fields and we launch our game from a screen.

Then we have the backend. Both the backend and GUI are part of the 'client desktop' component because both are present on the desktop of the user. The backend would include most code which isn't the GUI like authentication, game elements and movement. The GDX library already has some nice interfaces for game elements and movement, which we use in our game. This means the backend also uses interfaces from the GDX library.

We decided to not use a server-client approach to our architecture. We choose not to do this because we did not intent on making online multiplayer but only local multiplayer, so we concluded that the effort into making a server and client would not be worth it for us. We did not feel like there would be any substantial advantage having some code server-sided (apart from security reasons). All the features that we included or planned to be included could be done client-side.

We also intentionally choose the GDX library over other libraries since this library was specifically made to create games with. As already mentioned, a lot of our component are dependent on this library. We thought about using JavaFX for the GUI part, but since the GDX library already has GUI interfaces ready for use which are much more easily integrated into the rest of the library, we decided to only use the GDX library.

To conclude, although our architecture isn't very complex, there are definitely a few components which require each other to function. Especially the GDX library plays an major role in our game as without this library almost all game elements would have to be made by ourselves, which could lead to unrealistic physics, complicated classes that we would have to make and a more general GUI which does not suit the games aesthetics.


Database
account info
<<component>>
3rd party services
GDX library
physics engine
body elements
<<component>>
Client desktop
Backend
GUI
screens