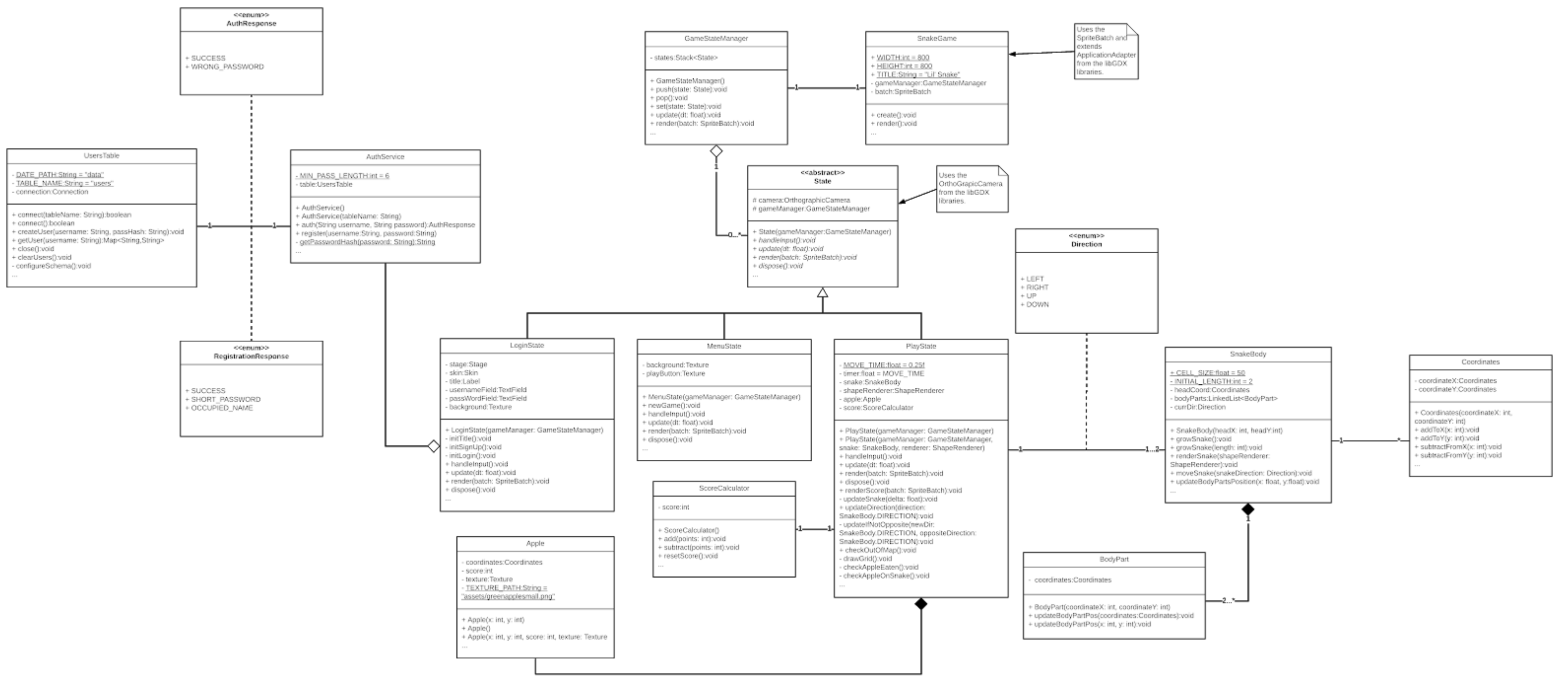
Assignment 2

Group 10

Exercise 1

The game starts in the SnakeGame class, which then calls a new GameStateManager. The GameStateManager has a stack of different states that can be invoked. The user can log in using the LoginState, where the AuthService checks your credentials and uses the database (UsersTable class is used here to do the db related stuff). Furthermore, there is the MenuState, PlayState and SignUpState (not included here because it is yet to be implemented and decided on how it will correlate with the database class, we might change the structure of how the database is later on).

The PlayState creates a new game and takes care of the logic of the game, such as the score (using the ScoreCalculator), the food (only Apple for now) and also the growth of the snake. The SnakeBody class keeps track of the snake movements using the Coordinates class and it uses BodyPart class to grow the snake whenever food is eaten.

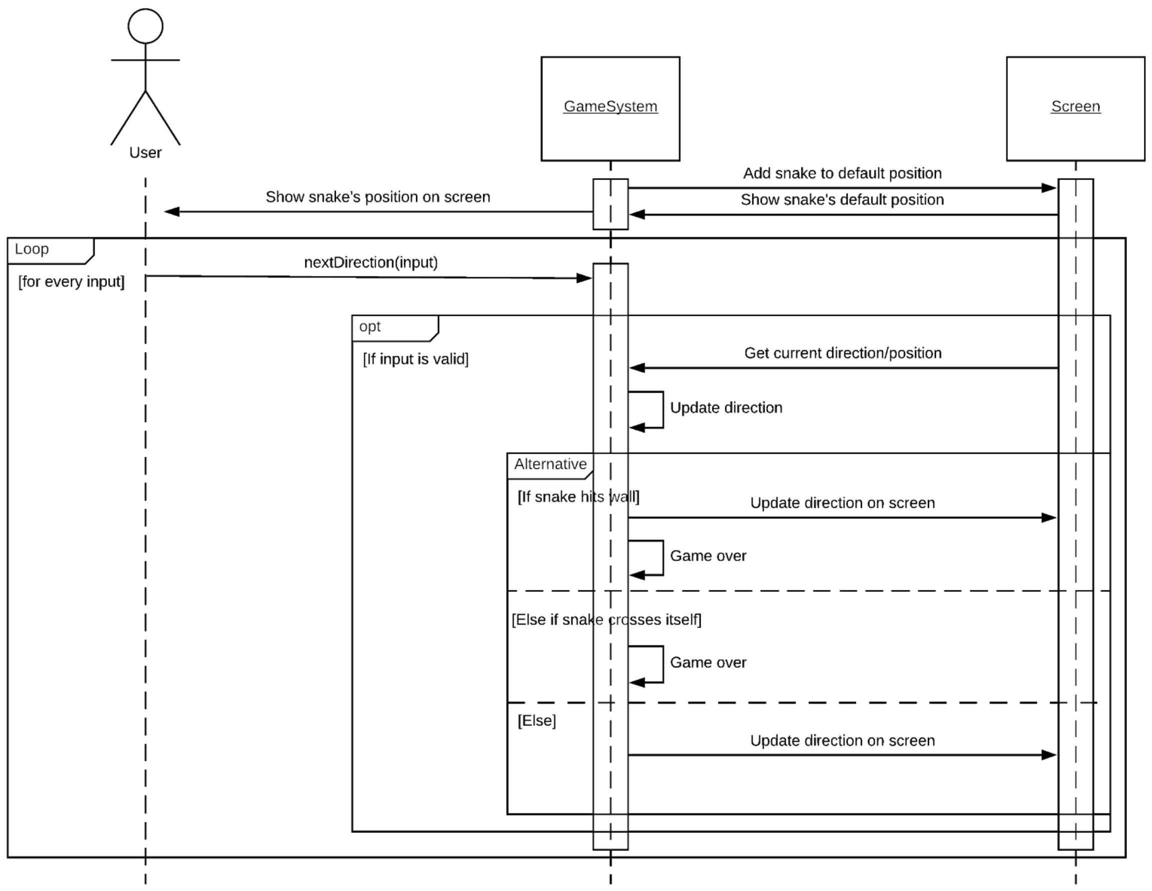


<<enum>> AuthResponse
+ SUCCESS
+ WRONG_PASSWORD
UsersTable
- DATE_PATH:String = "data"
- TABLE_NAME:String = "users"
- connection:Connection
+ connect(tableName: String):boolean
+ connect():boolean
+ createUser(username: String, passHash: String):void
+ getUser(username: String):Map<String,String>
+ close():void
+ clearUsers():void
- configureSchema():void
...
1
1
AuthService
- MIN_PASS_LENGTH:int = 6
- table:UsersTable
+ AuthService()
+ AuthService(tableName: String)
+ auth(String username, String password):AuthResponse
+ register(username:String, password:String)
- getPasswordHash(password: String):String
...
<<enum>> RegistrationResponse
+ SUCCESS
+ SHORT_PASSWORD
+ OCCUPIED_NAME
GameStateManager
- states:Stack<State>
+ GameStateManager()
+ push(state: State):void
+ pop():void
+ set(state: State):void
+ update(dt: float):void
+ render(batch: SpriteBatch):void
...
1
1
SnakeGame
+ WIDTH:int = 800
+ HEIGHT:int = 800
+ TITLE:String = "Lil' Snake"
- gameManager:GameStateManager
- batch:SpriteBatch
+ create():void
+ render():void
...
Uses the SpriteBatch and extends ApplicationAdapter from the libGDX libraries.
1
0...*
<<abstract>> State
# camera:OrthographicCamera
# gameManager:GameStateManager
+ State(gameManager:GameStateManager)
+ handleInput():void
+ update(dt: float):void
+ render(batch: SpriteBatch):void
+ dispose():void
...
Uses the OrthoGrapicCamera from the libGDX libraries.
<<enum>> Direction
+ LEFT
+ RIGHT
+ UP
+ DOWN
LoginState
- stage:Stage
- skin:Skin
- title:Label
- usernameField:TextField
- passWordField:TextField
- background:Texture
+ LoginState(gameManager: GameStateManager)
- initTitle():void
- initSignUp():void
- initLogin():void
+ handleInput():void
+ update(dt: float):void
+ render(batch: SpriteBatch):void
+ dispose():void
...
MenuState
- background:Texture
- playButton:Texture
+ MenuState(gameManager: GameStateManager)
+ newGame():void
+ handleInput():void
+ update(dt: float):void
+ render(batch: SpriteBatch):void
+ dispose():void
...
PlayState
- MOVE_TIME:float = 0.25f
- timer:float = MOVE_TIME
- snake:SnakeBody
- shapeRenderer:ShapeRenderer
- apple:Apple
- score:ScoreCalculator
+ PlayState(gameManager: GameStateManager)
+ PlayState(gameManager: GameStateManager, snake: SnakeBody, renderer: ShapeRenderer)
+ handleInput():void
+ update(dt: float):void
+ render(batch: SpriteBatch):void
+ dispose():void
+ renderScore(batch: SpriteBatch):void
- updateSnake(delta: float):void
+ updateDirection(direction: SnakeBody.DIRECTION):void
- updateIfNotOpposite(newDir: SnakeBody.DIRECTION, oppositeDirection: SnakeBody.DIRECTION):void
+ checkOutOfMap():void
- drawGrid():void
- checkAppleEaten():void
- checkAppleOnSnake():void
...
ScoreCalculator
- score:int
+ ScoreCalculator()
+ add(points: int):void
+ subtract(points: int):void
+ resetScore():void
...
1
1
Apple
- coordinates:Coordinates
- score:int
- texture:Texture
- TEXTURE_PATH:String = "assets/greenapplesmall.png"
+ Apple(x: int, y: int)
+ Apple()
+ Apple(x: int, y: int, score: int, texture: Texture
...
1
1...2
SnakeBody
+ CELL_SIZE:float = 50
- INITIAL_LENGTH:int = 2
- headCoord:Coordinates
- bodyParts:LinkedList<BodyPart>
- currDir:Direction
+ SnakeBody(headX: int, headY:int)
+ growSnake():void
+ growSnake(length: int):void
+ renderSnake(shapeRenderer: ShapeRenderer):void
+ moveSnake(snakeDirection: Direction):void
+ updateBodyPartsPosition(x: float, y:float):void
...
1
*
Coordinates
- coordinateX:Coordinates
- coordinateY:Coordinates
+ Coordinates(coordinateX: int, coordinateY: int)
+ addToX(x: int):void
+ addToY(y: int):void
+ subtractFromX(x: int):void
+ subtractFromY(y: int):void
...
1
2...*
BodyPart
- coordinates:Coordinates
+ BodyPart(coordinateX: int, coordinateY: int)
+ updateBodyPartPos(coordinates:Coordinates):void
+ updateBodyPartPos(x: int, y: int):void

Link to full-sized UML class diagram here: https://bit.ly/2Pjq0WX

Exercise 2

This is our sequence diagram, it shows the interaction between the user and the snake when the game has just started and the user can make the snake move. Initially the GameSystem adds the snake's default position to the screen, and the screen shows the snake's position. Then a loop begins where, the user can click WASD to control the snake's movement. Within that loop every input gets checked if it's valid. And if it is the direction gets updated accordingly. If the snake hits the wall or crosses itself the game ends, otherwise the direction gets updated and the system will continue to check for the next input again.


User
GameSystem
Screen
Add snake to default position
Show snake's position on screen
Show snake's default position
Loop
[for every input]
nextDirection(input)
opt
[If input is valid]
Get current direction/position
Update direction
Alternative
[If snake hits wall]
Update direction on screen
Game over
[Else if snake crosses itself]
Game over
[Else]
Update direction on screen

Link to full-sized UML sequence diagram here: shorturl.at/loCDE

Exercise 3

With approval from our TA Sara, for the implementation part of this assignment, we made sure to do the following:

- Set up authentication and database for it. Only login possible from the user side.
- Pick a game name
- Have an authentication screen UI (signup not working)
- Render one food element on screen
- Fix Checkstyle errors we had from previous sprint