Class Diagrams

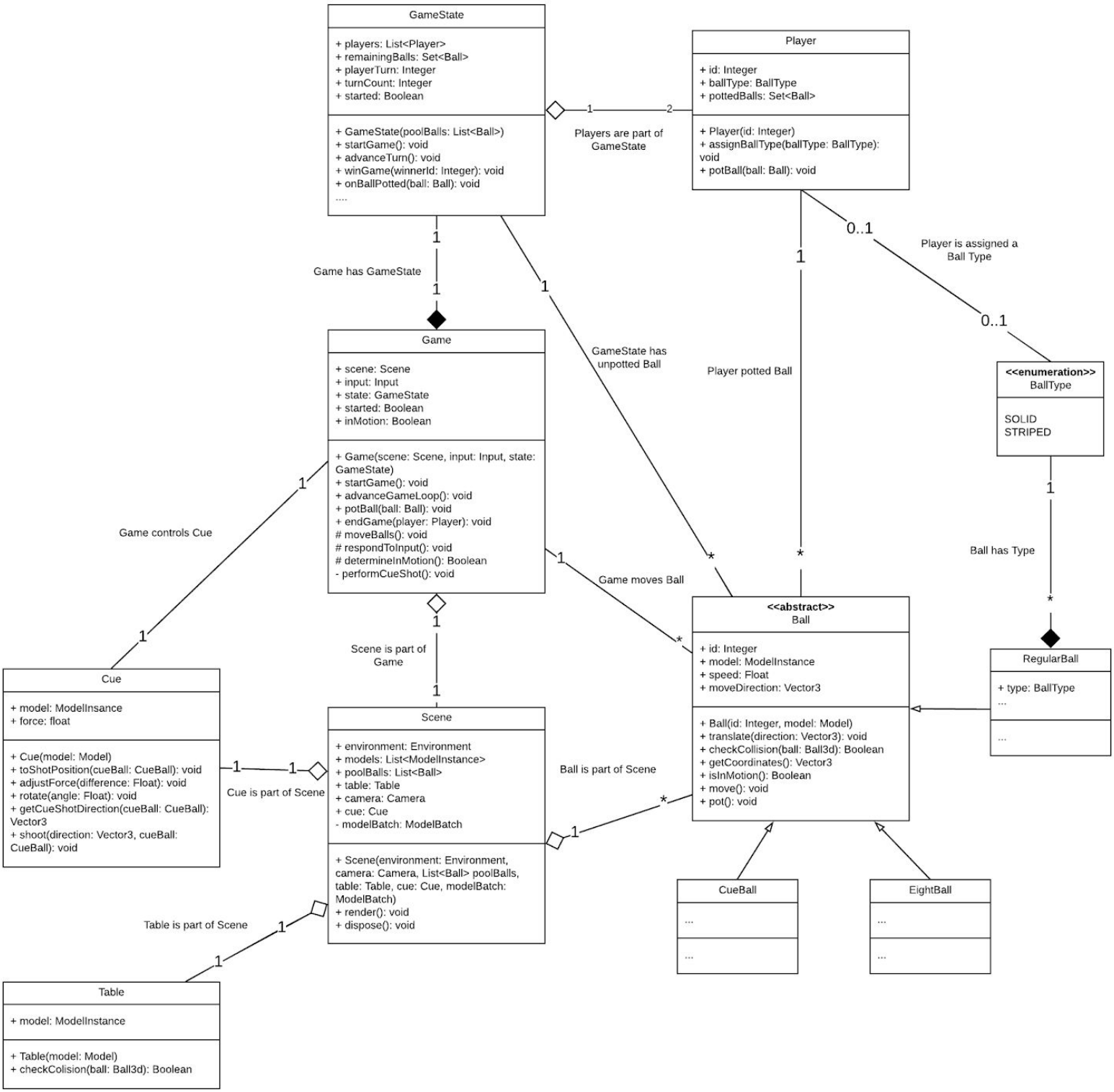
First page diagrams (next page)
Since the objective in Pool is to pot all the balls, the Ball is the core element of the game. Thus, the Ball class was added to the UML diagram with all the possible ball types of classes (which are all important in a game of Pool). We separated the different Ball Types into different classes for easier extension of the game logic to make the structure of the game cleaner in terms of handling ball-specific functionality (potting, determining loss of the game, etc.)

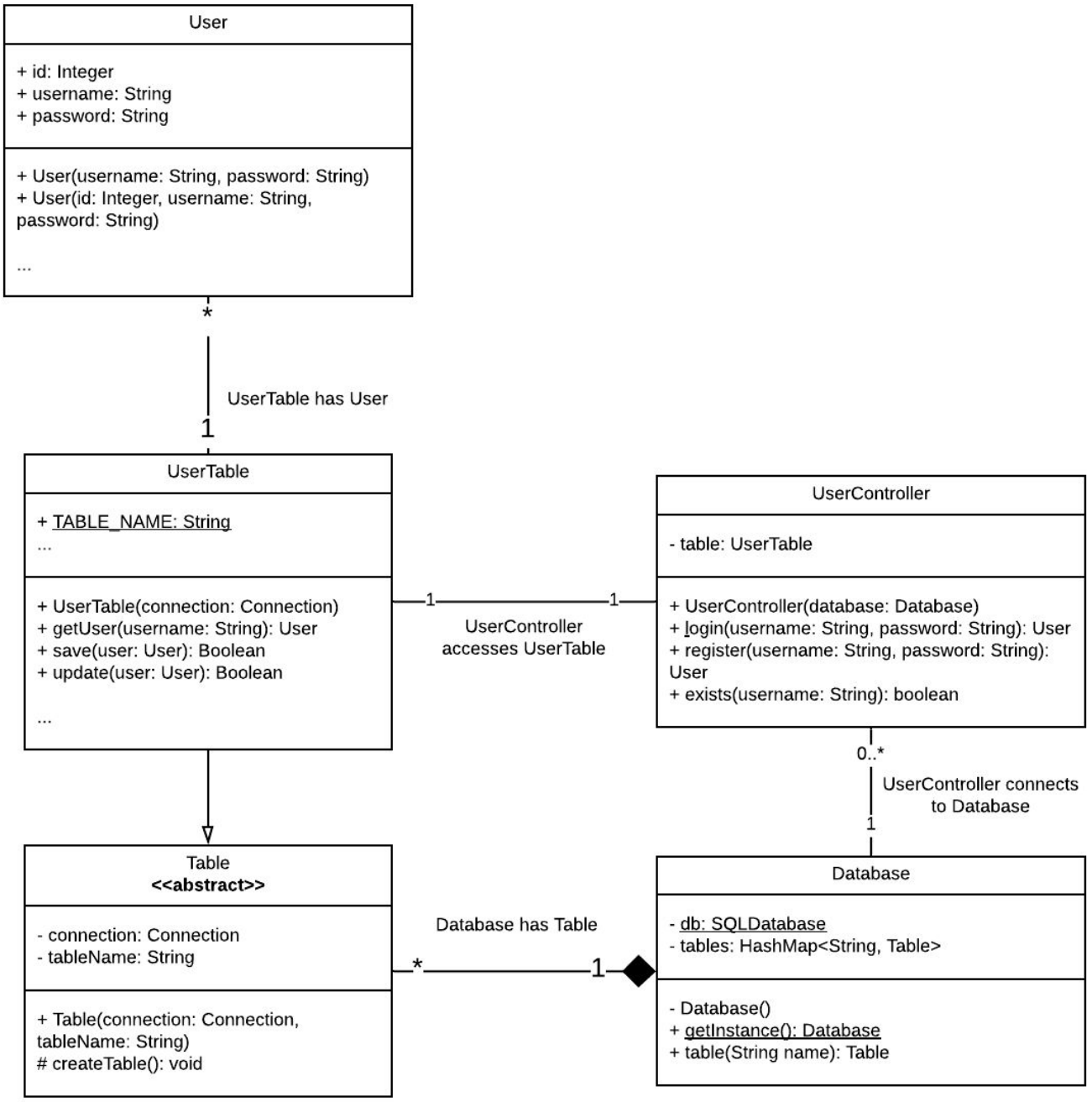
The Player class was added due to the Player being a participant in the Game. The reason we added it as a separate class is to keep track of Player-specific variables easier, namely the ID and the balls they have potted. The GameState class was added because it handles the logical flow of the Game. This includes advancing the turns of the Players, handling potting balls on the logical side of the game, handling victory & loss, as well as keeping track of relevant game variables (namely the balls remaining to be potted, and the Players of the game)

The Game class handles the actual input from the user as well as the game loop for each iteration, hence it is one of the core classes of the game. In addition, it also holds the internal GameState which handles the logic. The purpose of the Game class is to encapsulate Game-specific functionality (getting input, moving balls every iteration, triggering collisions every iteration, and so on) from the actual logic of the game (which is handled by GameState).

The Scene class was chosen because it provides access to all the game elements of the game, and handles the rendering of the 3D models corresponding to the game elements. The Table class was added because it represents the actual table of the game, which is a core game element of Pool. The Cue itself holds the 3D model of the Cue, and has functionality related to controlling the Cue's parameters (power and angle), and is also a core part of the Pool game as well as the Scene (as a 3D model and as a game element), since it holds functionality required to perform the cue shot.

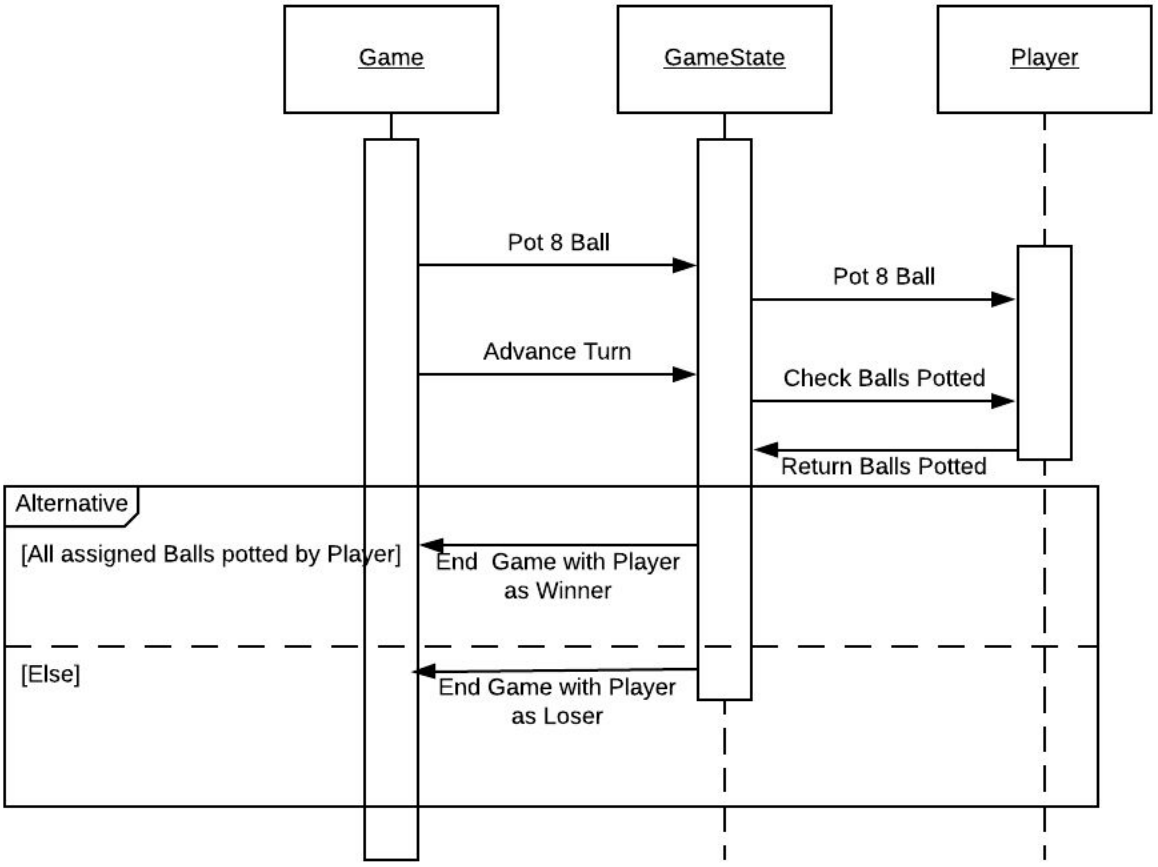
Second page diagrams
The User class was chosen since it represents User objects stored in the database. The Table and UserTable classes correspond to SQL database tables in the database, hence they are very important core parts of the game, as they cover the data storage for authentication. The Database class was chosen since it handles holding the actual SQL database, and abstracts the creation of it away from data accessing classes, as well as provides an interface to access the tables. Finally, the UserController provides a simplified way of accessing the User table to make the interface more transparent with regards to authentication.


GameState
+ players: List<Player>
+ remainingBalls: Set<Ball>
+ playerTurn: Integer
+ turnCount: Integer
+ started: Boolean
+ GameState(poolBalls: List<Ball>)
+ startGame(): void
+ advanceTurn(): void
+ winGame(winnerId: Integer): void
+ onBallPotted(ball: Ball): void
....
Player
+ id: Integer
+ ballType: BallType
+ pottedBalls: Set<Ball>
+ Player(id: Integer)
+ assignBallType(ballType: BallType): void
+ potBall(ball: Ball): void
1
2
Players are part of GameState
1
Game has GameState
1
1
GameState has unpotted Ball
1
Player potted Ball
0..1
Player is assigned a Ball Type
0..1
Game
+ scene: Scene
+ input: Input
+ state: GameState
+ started: Boolean
+ inMotion: Boolean
+ Game(scene: Scene, input: Input, state: GameState)
+ startGame(): void
+ advanceGameLoop(): void
+ potBall(ball: Ball): void
+ endGame(player: Player): void
# moveBalls(): void
# respondToInput(): void
# determineInMotion(): Boolean
- performCueShot(): void
<<enumeration>>
BallType
SOLID
STRIPED
1
Ball has Type
*
1
Game controls Cue
1
1
Game moves Ball
*
*
<<abstract>>
Ball
+ id: Integer
+ model: ModelInstance
+ speed: Float
+ moveDirection: Vector3
+ Ball(id: Integer, model: Model)
+ translate(direction: Vector3): void
+ checkCollision(ball: Ball3d): Boolean
+ getCoordinates(): Vector3
+ isInMotion(): Boolean
+ move(): void
+ pot(): void
RegularBall
+ type: BallType
...
...
1
Scene is part of Game
1
Cue
+ model: ModelInsance
+ force: float
+ Cue(model: Model)
+ toShotPosition(cueBall: CueBall): void
+ adjustForce(difference: Float): void
+ rotate(angle: Float): void
+ getCueShotDirection(cueBall: CueBall): Vector3
+ shoot(direction: Vector3, cueBall: CueBall): void
1
1
Cue is part of Scene
Scene
+ environment: Environment
+ models: List<ModelInstance>
+ poolBalls: List<Ball>
+ table: Table
+ camera: Camera
+ cue: Cue
- modelBatch: ModelBatch
+ Scene(environment: Environment, camera: Camera, List<Ball> poolBalls, table: Table, cue: Cue, modelBatch: ModelBatch)
+ render(): void
+ dispose(): void
Ball is part of Scene
*
1
CueBall
...
...
EightBall
...
...
Table is part of Scene
1
1
Table
+ model: ModelInstance
+ Table(model: Model)
+ checkColision(ball: Ball3d): Boolean


User
+ id: Integer
+ username: String
+ password: String
+ User(username: String, password: String)
+ User(id: Integer, username: String, password: String)
...
*
UserTable has User
1
UserTable
+ TABLE_NAME: String
...
+ UserTable(connection: Connection)
+ getUser(username: String): User
+ save(user: User): Boolean
+ update(user: User): Boolean
...
1
1
UserController accesses UserTable
UserController
- table: UserTable
+ UserController(database: Database)
+ login(username: String, password: String): User
+ register(username: String, password: String): User
+ exists(username: String): boolean
0..*
UserController connects to Database
1
Table
<<abstract>>
- connection: Connection
- tableName: String
+ Table(connection: Connection, tableName: String)
# createTable(): void
Database has Table
*
1
Database
- db: SQLDatabase
- tables: HashMap<String, Table>
- Database()
+ getInstance(): Database
+ table(String name): Table

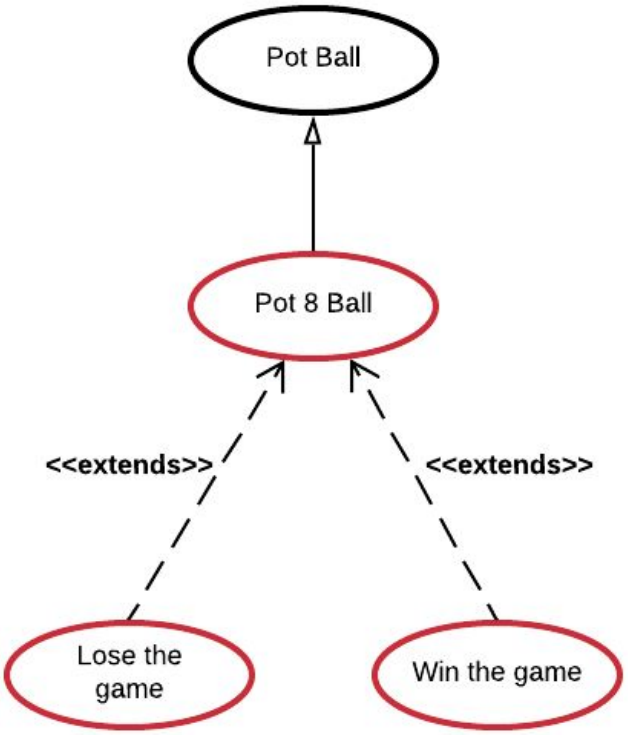
Sequence Diagrams

Sequence Diagram #1: Potting 8-Ball

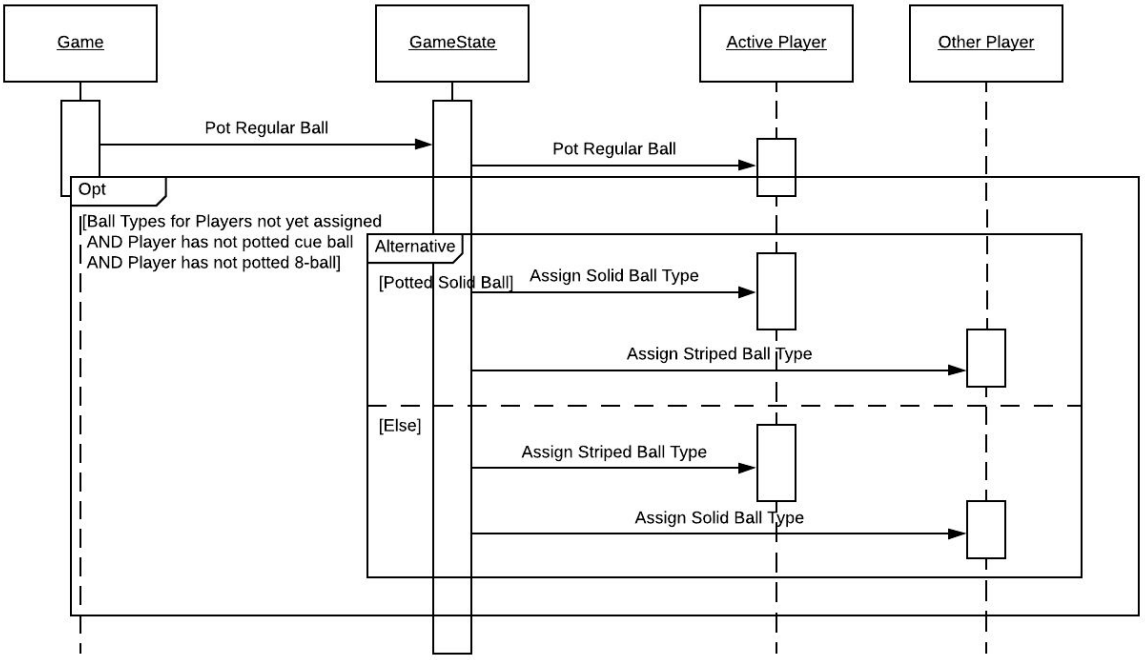


Use Cases Covered

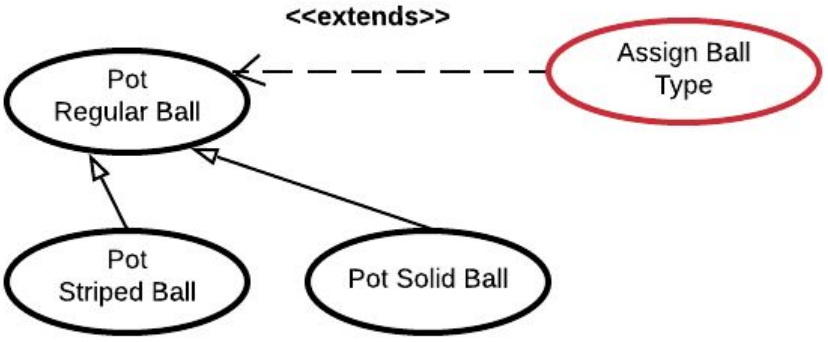
Covers the following use cases:



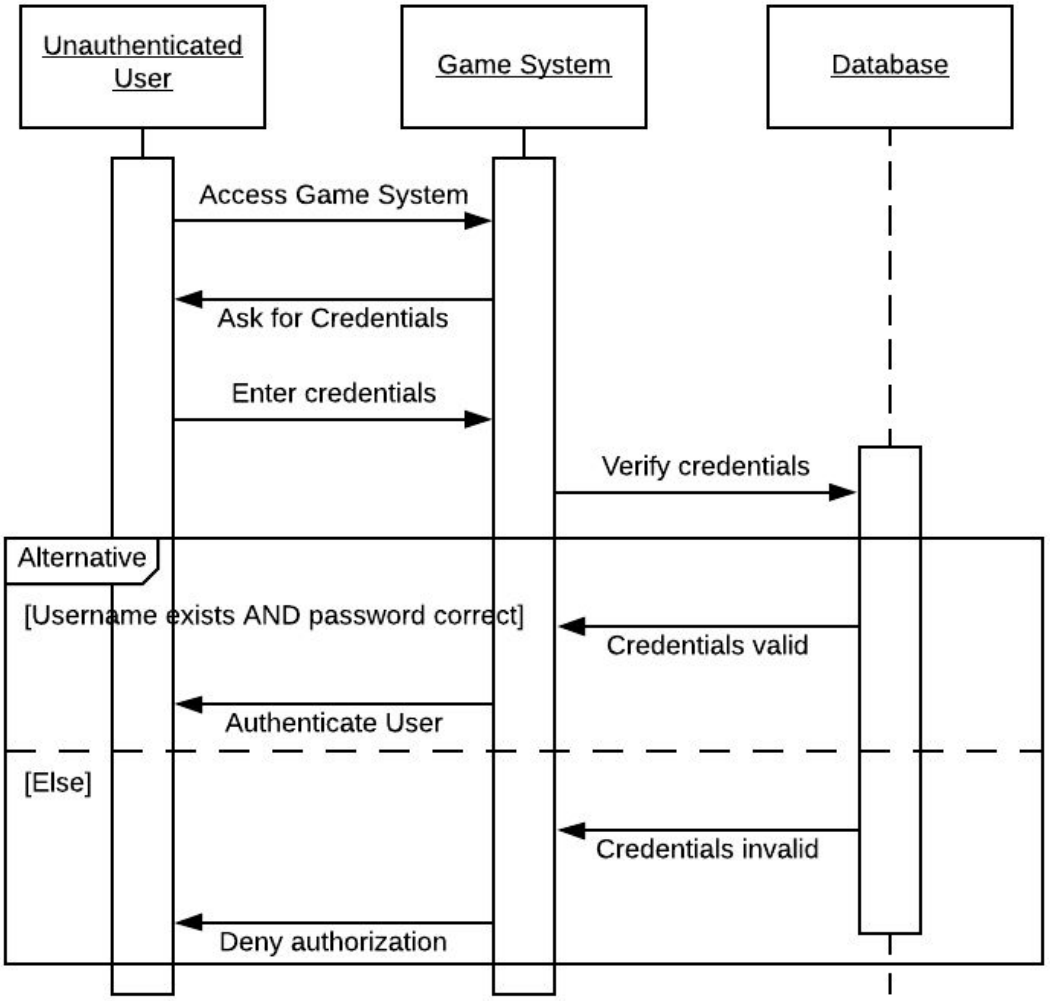
Sequence Diagram #2: Ball Type Assignment



Use Cases Covered



Sequence Diagram #3: Authentication



Use Cases Covered