Final

Shawarma

Contents

Introduction

Bricking Bad is a fun and challenging game that has been developed by team Shawarma as part of COMP302 course. The game involves bricks and aliens. The player has to destroy all bricks to win the game, at the time when the aliens stand in the face of the player winning the game. Additionally, there exists a set of powerups which can help the player achieve better performance and win the game faster.

Bricking Bad was designed by following software design principles, as well as implementing various design patterns, making the design and implementation as modular and reusable as possible. Also, several optimizations have been used to boost the performance, and minimize the data traffic across different parts of the design.

This report shall go over all the design stages, diagrams, use cases, and patterns used on the way of designing this game.

Teamwork Organization

Designing and programming a game through the semester is different than any assignment of the lectures in terms of the time it takes and the development cycle. To prevent any miscommunication, we held weekly team meetings where each member comes with questions in mind and tries to find answers to these questions together. Especially design discussions of the project is the most critical part in terms of communication since any mistake can cause waste of time.

In the first weeks of the project, all team members worked on requirements together since it was important to be on the same page. After we decided on the class diagram and the design patterns we would use, each member got specific parts of the game until the first demo. However, each team member explicitly works on each part of the project from the user interface to unit tests. We mostly shared open issues and brought solutions to them. Since each team member needs to experience each step of the agile methodologies, we always paid attention to agile principles as possible as we could.

The busiest time interval for the project was a few weeks just before the final demonstration since the project should be finished. We met more frequently in these weeks and try to complete the project without any unimplemented features. In these weeks, the number of branches increased rapidly since we were implementing new features each day.

In the last week, we mostly fixed bugs we found while playing the game. All members of the team gained invaluable experience during the project.

Use Cases

Use Case UC1: Access Help Screen

Scope: The Game

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to reach help information

Preconditions: Running Application

Postconditions: User welcomed with a help screen

Main Success Scenario:

1. In the login screen, the user wants to access the help screen
2. The user clicks the help button
3. The help screen is shown to the user

Extensions:

- *a. The program fails.
    1. The user runs the game again.

Special Requirements:

- The Help screen should be readable and easy to understand

Technology and data variations:

- Provide the help screen in multiple languages
- Include a text version, and a visual version (provide video guide)

Frequency of occurence: Whenever the users wishes to seek help

---

Use Case UC2: Activate a Manual Power-Up

Scope: Gameplay

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to activate a manual power-up in their inventory

Preconditions: User has the power-up in their inventory.

Postconditions: The power-up is active

Main Success Scenario:

1. User gets a power-up
2. User clicks the power-up icon or presses the appropriate key
3. The game removes the power-up from the inventory
4. The game activates the power-up

Extensions:

- *a. The program fails
  - 1. The user runs the game again.
- *2a. The user clicks an icon/types a letter corresponding to a power-up they don't have
  - 1. The power-u doesn't activate
  - 2. The game continues without change

Special Requirements:

- The power-up icons are in the user's inventory. A power-up icon is transparent if the user doesn't have the power-up, solid they have it.

Technology and data variations:

- Power-up can be either activated using mouse or keyboard.

Frequency of occurence: The number of times a manual power-up is activated is either equal to or less than the number times manual power-ups are acquaired.

Use Case UC3: Authenticate

Scope: Game login screen

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to authenticate himself so that he can access his saved data and play the game

Preconditions: The game is initialized and on the login screen

Postconditions: The user is authenticated and has access to the game

Main Success Scenario:

1. User provides his username and password, chooses to continue
2. Game authenticates the User and shows a welcome message
3. User presses continue
4. Game changes the board to the building mode view

Extensions:

- *2a. The username or password is incorrect
  - 1. The Game informs the user that the login credentials are incorrect
  - 2. User enters the correct information
  - 3. Game authenticates the user and shows a welcome message
  - 4. User presses continue
  - 5. Game changes the board to the building mode view

Special Requirements:

- Password should be hidden from view
- Care should be taken when handling the password for security reasons

Technology and data variations:

- Keyboard might have caps lock on or might be in another language

Frequency of occurence: Once per game session

Use Case UC4: Break a Brick

Scope: Gameplay

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to destroy all the blocks in order to win the game

Preconditions: User is playing.

Postconditions: Brick is broken and removed from Board

Main Success Scenario:

1. Ball is moving around on the screen and heads towards the bottom of the screen
2. User moves the paddle to be directly below the ball: Include Move the Paddle
3. The ball hits the paddle
4. The Game changes the trajectory of the ball
5. The ball hits a brick
6. The Game removes the brick

Extensions:

- *a. The program fails.
  - 1. The user runs the game again.
- *3a. The paddle misses the ball and the balls falls off screen
  - 1. The player loses a life
  - 2. Game shows another ball (if the use still has a life) and the game continues
- *4a. The ball misses all bricks
  - 1. The ball reflects back to the player and we go back to step 1 in the main scenario
- *5a. The ball is a half-metal brick type and the ball hits it from the metal side
  - 1. The ball reflects back normally and the brick is not destroyed

Special Requirements:

- none

Technology and data variations:

- none

Frequency of occurence: Nearly continuous

Use Case UC5: Build a Map

Scope: Building mode

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to create a new map that can be loaded in future in order to play the game

Preconditions: User has a valid account with a username and password

Postconditions: The user-created map appears in the map list

Main Success Scenario:

1. The user enters building mode by clicking the “building mode” button
2. The user interacts with the Game to specify the number of each brick type
3. The Game creates bricks on random places
4. The user moves the randomly created bricks into new positions if s/he wants
5. The user saves the map if the minimum requirements are met
6. The map is saved

Extensions:

- *a. The program fails
  - 1. The User runs the game again and starts over
- 2a. The numbers don’t meet the requirements
  - 1. Change numbers to satisfy requirements
- 3a. The display cannot contains the number of bricks specified by the user
  - 1. The Game warns the user to decrease the number of bricks.
- 4a. The user tries to place a brick such it overlaps with another brick
  - 1. The Game warns the user to place the brick into another place where it doesn’t overlap with a brick.
- 5a. The minimum requirements are not met
  - 1. The Game warns the user to satisfy minimum requirements
- 6a. The disk is full so the map cannot be saved.
  - 1. Delete another map or increase the disk capacity

Special Requirements:

- The buttons and bricks on the screen should be visible
- Write permission for storage access

Technology and data variations:

- Provide different color schemes for color blind people
- Different storage devices such as cloud, local etc.

Frequency of occurence: At User’s demand

Use Case UC6: Make an Account

Scope: Game login screen

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to have an account that they can use in order to play the game

Preconditions: User does not have an account and is on the login screen

Postconditions: User has a valid account with a username and password

Main Success Scenario:

1. User chooses the option of making a new account
2. The Game displays the account registration screen
3. User provides a username and password to the fields
4. Game creates an account for the user and informs the user that the account has been created successfully.

Extensions:

- *a. The program fails
  - 1. The User runs the game again and starts over
- 3a. The username is not valid
  - 1. The Game informs the user that the username entered is not valid
  - 2. User enters a valid username
- 3b. The password is not valid
  - 1. The Game informs the user that the username entered is not valid
  - 2. User enters a valid username
  - 3. Game creates an account for the user and informs the user that the account has been created successfully
- 4a. Another user with the chosen username already exists.
  - 1. The Game informs the user that a user with the chosen username already exists and displays login screen

Special Requirements:

- The password entered by the user should not be visible on the screen

Technology and data variations:

- Keyboard of the user might give input in varying languages

Frequency of occurence: Once per user

Use Case UC7: Fire Destructive Laser Gun

Scope: Gameplay

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: wants to destroy a brick using the destructive laser gun power-up

Preconditions: The Destructive Laser Gun is activated

Postconditions: There laser is fired

Main Success Scenario:

1. User presses the appropriate key or clicks.
2. The laser gun fires
3. The game decrements number of laser gun shots remaining by 1.
4. If the gun hits a brick, it gets destroyed.

Extensions:

- *3a. The shot fired was the last shot of the laser.
  - 1. The gun at the ends of the paddle dissappears.
  - 2. The Destructive Laser Gun becomes inactive, user can't fire again.

Special Requirements:

- The number of shots remaining are displayed above the laser gun.

Technology and data variations:

- Either mouse or keyboard can be used to fire the laser gun

Frequency of occurence: Destructive Laser Gun can be fired at most 5 times after being activated once.

Use Case UC8: Get a Power-Up

Scope: Gameplay

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to acquire the power-up to be able to use it later in the game

Preconditions: User is playing the game

Postconditions: User has a power-up in his inventory

Main Success Scenario:

1. User breaks a wrapper brick that contains a power-up
2. Game releases power-up from broken wrapper brick and makes it fall.
3. User moves the paddle to be directly below the power up: include Move The Paddle
4. Power-up touches the paddle.
5. Game removes the power-up and gives it to the User

Extensions:

- *a. The program fails
  - 1. The user runs the game again.
- *3a. The paddle misses the power-up
  - 1. The player doesn't get the power-up
  - 2. The game continues
- *5a. The power-up user gets is a manual power-up
  - 1. The power-up is added to the user's inventory
  - 2. The power-up is automatically activated.
- *5b. The automatic power-up is Destructive Laser Gun
  - 1. The power-up is automatically activated.
  - 2. A laser gun appears at the both ends of the paddle
  - 3. The Destructive Laser Gun power-up is active.
- *5c. The automatic power-up is Fireball
  - 1. The power-up is automatically activated.
  - 2. The ball changes to a fireball
  - 3. The fireball damages also the bricks next to one it hits
  - 4. The fireball can destroy metal sides of bricks in two hits.

  - 5. The fireball return to normal when the user loses it.
- *5d. The automatic power-up is Gang-of-balls
  - 1. The power-up is automatically activated.
  - 2. After the ball hits the paddle, it multiplies by 10.

Special Requirements:

- Any acquired manual power-ups are displayed at the inventory using icons.
- If the 10 balls created by the Gang-of-balls power-up move with the same speed, but with an angle equals to the ball index multiplied by 360 and divided by 10.

Technology and data variations:

- A keyboard is used to move the paddle.

Frequency of occurence: In a game session, a possibility every time a wrapper brick is broken.

Use Case UC9: Hit Harmful Alien

Scope: Gameplay

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to hit harmful aliens so that they disappear from screen.

Preconditions: At least one harmful alien appears in game screen.

Postconditions: At least one harmful alien has disappeared

Main Success Scenario:

1. User performs **Move the Paddle** in order to direct the ball towards the harmful alien
2. Game shows the user the movement of the ball on the board.
3. User directs the ball towards the harmful alien.
4. Game shows Harmful Alien is hit by the ball in the way that causes it to disappear.
5. Game removes Harmful Alien from board.

Extensions:

- *a. The program fails
  - 1. User runs the game again.
- *4a. The ball does not hit any harmful alien.
  - 1. User performs **Move the Paddle** again to hit alien.
- *4b. Harmful Alien was not hit in the proper way that makes it disappear.
  - 1. User performs **Move the Paddle** again to hit the alien

Special Requirements:

- none

Technology and data variations:

- none

Frequency of occurence: Throughout the game.

Use Case UC10: Load Saved Game

Scope: The Game

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to continue a previously saved game

Preconditions: User has an account and at least one game saved by the same User

Postconditions: Game is resumed from the saved state.

Main Success Scenario:

1. User opens the "Load Game" menu
2. User picks one of the previously saved sessions
3. The game loads that session
4. The User can press "Resume" and plays the game.

Extensions:

- *a. The program fails.
    1. The user runs the game again.
- 3b. Session File is corrupt:
    1. Game refuses to continue loading
    2. Game returns to "Load Game" menu

Special Requirements:

- Access to disk space (Read permission)

Technology and data variations:

- Different storage devices (cloud, local, . . .)

Frequency of occurence: Whenever the users wishes to

Use Case UC11: Move the Paddle

Scope: Gameplay

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to move the paddle in order to hit the ball or catch the falling power-up

Preconditions: User is playing

Postconditions: The position of the paddle has changed

Main Success Scenario:

1. The user predicts where s/he should move the paddle
2. The user moves the paddle by either pressing and releasing responsible buttons or keeping them down to move further

3. User can rotate the paddle by pressing required buttons up to {45, 135} degrees
4. The paddle moves according to input of the user
5. The paddle stops at the final location

Extensions:

- *a. The program fails.
  - 1. The user runs the game again
- *3a. The paddle stops
  - 1. The paddle hits the border of the game window so user can't move the paddle further

Special Requirements:

- A working keyboard is needed

Technology and data variations:

- none

Frequency of occurence: Nearly continuous

Use Case UC12: Pause the Game

Scope: Gameplay

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to pause the game in order to resume later

Preconditions: A game is in progress

Postconditions: The game is paused

Main Success Scenario:

1. User clicks on the pause button or presses the pause shortcut on the keyboard
2. Game halts the game and displays the pause screen
3. Game changes the pause button to a resume button

Extensions:

- *a. The program fails
  - 1. The User runs the game again and starts over

Special Requirements:

- The pause button on screen should be visible and easily identifiable as a pause button
- The button should be easily accessible on the keyboard

Technology and data variations:

- Different keyboard layouts to keep in mind

Frequency of occurence: Multiple times per game

Use Case UC13: Quit the Game

Scope: The Game

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to quit the game

Preconditions: A game is in progress

Postconditions: The game is closed

Main Success Scenario:

1. User clicks on the quit button or shortcut on the keyboard
2. Game halts the game and displays a message to be confirmed by user
3. User confirms quitting
4. Game closes the game

Extensions:

- *a. The program fails
  - 1. The game is already closed, reached post condition

Special Requirements:

- The quit button on screen should be visible and easily identifiable
- The button should be easily accessible on the keyboard

Technology and data variations:

- Different keyboard layouts to keep in mind

Frequency of occurence: Once per game

Use Case UC14: Release Magnetized Ball

Scope: Gameplay

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: wants to release the ball captured by the magnetized paddle

Preconditions: The Magnetized Ball power-up is active

Postconditions: The ball is released

Main Success Scenario:

1. The ball touches the paddle, Game stops it and makes it stuck to the paddle
2. The user presses the appropriate key or clicks
3. The Game releases the ball from the paddle
4. The Game deactivaes the Magnet power-up

Extensions:

- *1a. The paddle misses the ball
  - 1. The power-up is lost.

Special Requirements:

- After being relesead from the paddle, the ball preserves it's previous speed and direction.

Technology and data variations:

- Either keyboard or mouse can be used to release the ball.

Frequency of occurence: Happens once after every activation of a magnet power-up

---

Use Case UC15: Resume the Game

Scope: The Game

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to continue the game from pause

Preconditions: The game is paused

Postconditions: The game is in progress

Main Success Scenario:

1. User clicks on the resume button or presses the resume shortcut on the keyboard
2. Game continues the game and displays the removes the pause screen
3. Game changes the resume button to a pause button

Extensions:

- *a. The program fails
  - 1. The User runs the game again and starts over

Special Requirements:

- There should be a short delay after choosing resume in order for the user to get ready.
- The resume button on screen should be visible and easily identifiable as a resume button
- The button should be easily accessible on the keyboard

Technology and data variations:

- Different keyboard layouts to keep in mind

Frequency of occurence: Multiple times per game

Use Case UC16: Save the Game

Scope: The Game

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to save the state of game to resume at the same state later

Preconditions: The game is paused.

Postconditions: The game is saved according to its state at the moment of pausing.

Main Success Scenario:

1. Game shows the user the option menu.
2. User chooses the save option.
3. Game prompts user to enter a save name identifying the saved state of the game.
4. User enters a save name to identify the save.
5. Game saves the state of the game and links it with the save name the user entered.
6. Game shows the User that the operation is successful.

Extensions:

- *a. The program fails
  - 1. User runs the game again.
- *3a. The storage disk is full.
  - 1. Game shows the user that there is no enough storage after choosing save option.
  - 2. User frees storage and retries to save steps.
- *5a. Save name entered by user already exists as a save name for a previously saved game.
  - 1. Game asks the user if he wants to overwrite this save name or choose another save name.
  - 2. User either chooses to overwrite, in which case previous save information is lost, or enter another save name.

Special Requirements:

- none

Technology and data variations:

- none

Frequency of occurence: At User's demand.

Use Case UC17: Change Storage Provider

Scope: The Game

Level: user goal

Primary Actor: User

Stakeholders and interests:

- User: Wants to change the storage provider

Preconditions: The game is turned off.

Postconditions: The next time the game is turned on, it will use the specified storage provider.

Main Success Scenario:

1. User turns off the game
2. User opens game.properties file
3. User changes Bin to MapDB
4. User turns on the game

Extensions:

- *a. The program fails
  - 1. User runs the game again.

Special Requirements:

- none

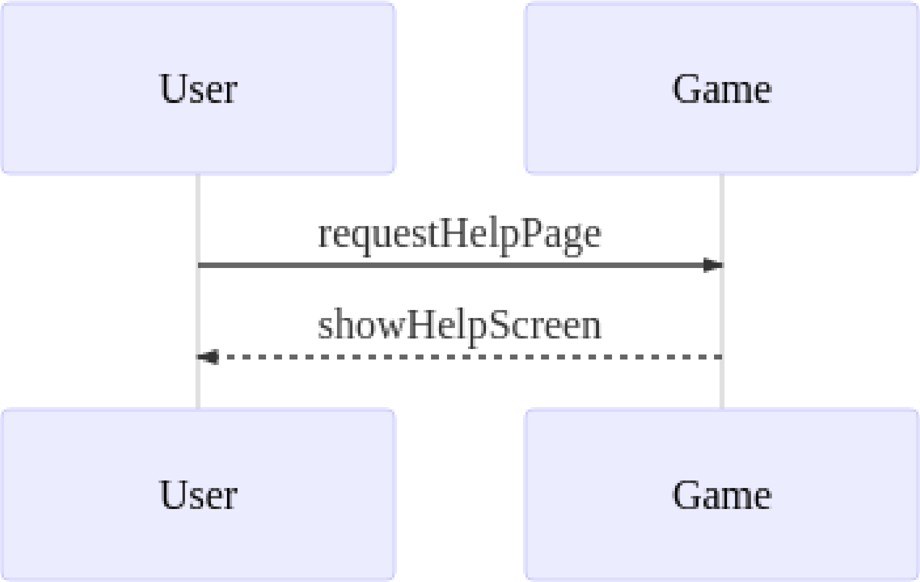
Technology and data variations:

- none

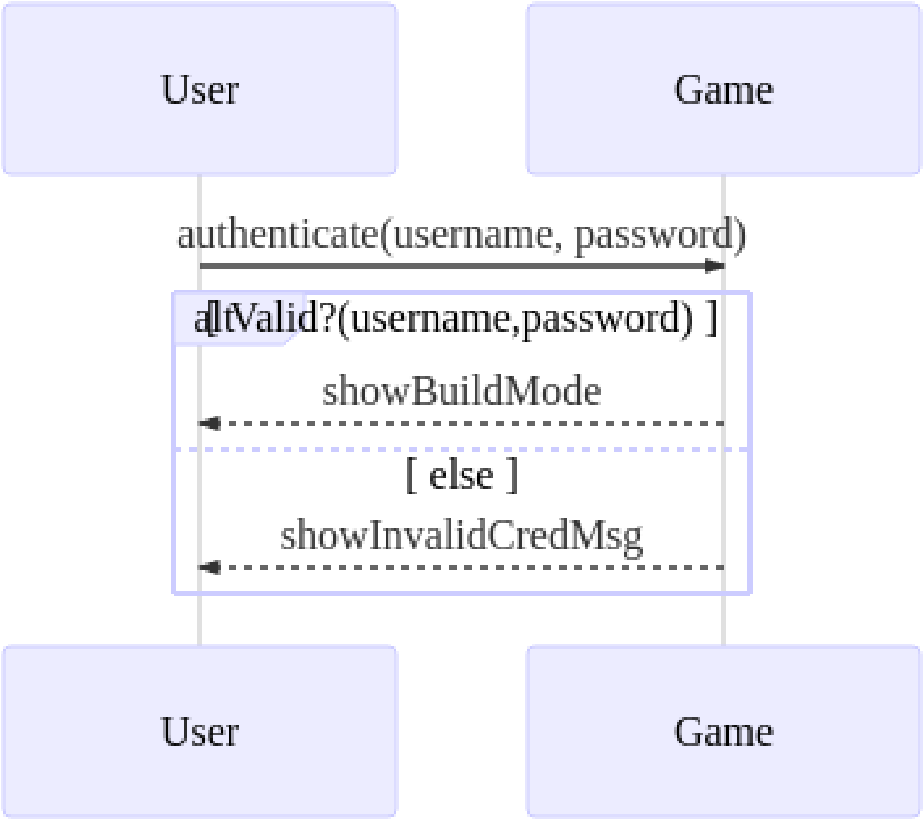
Frequency of occurence: At User's demand.

System Sequence Diagrams

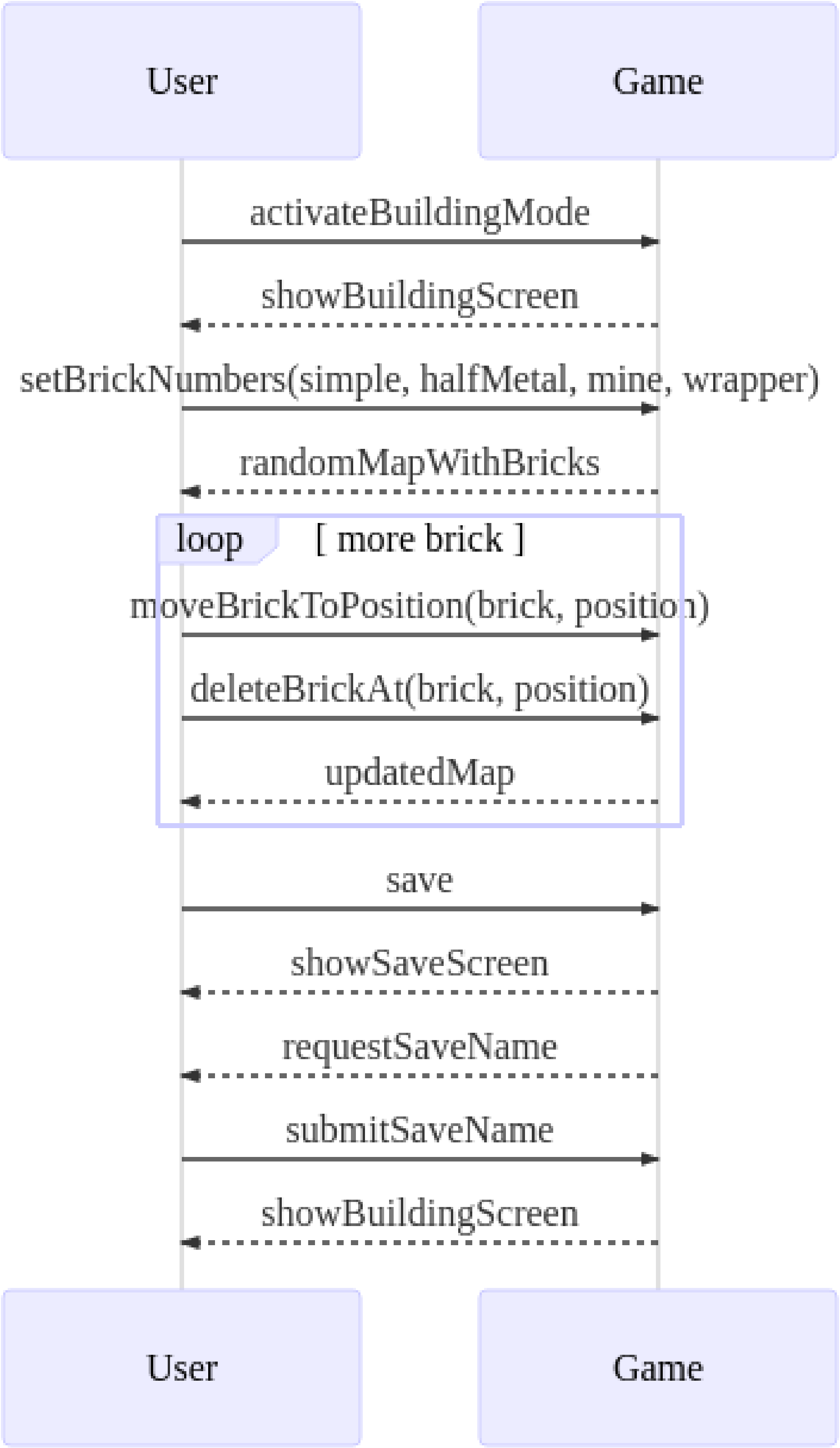
Access help screen Game Sequence Diagram



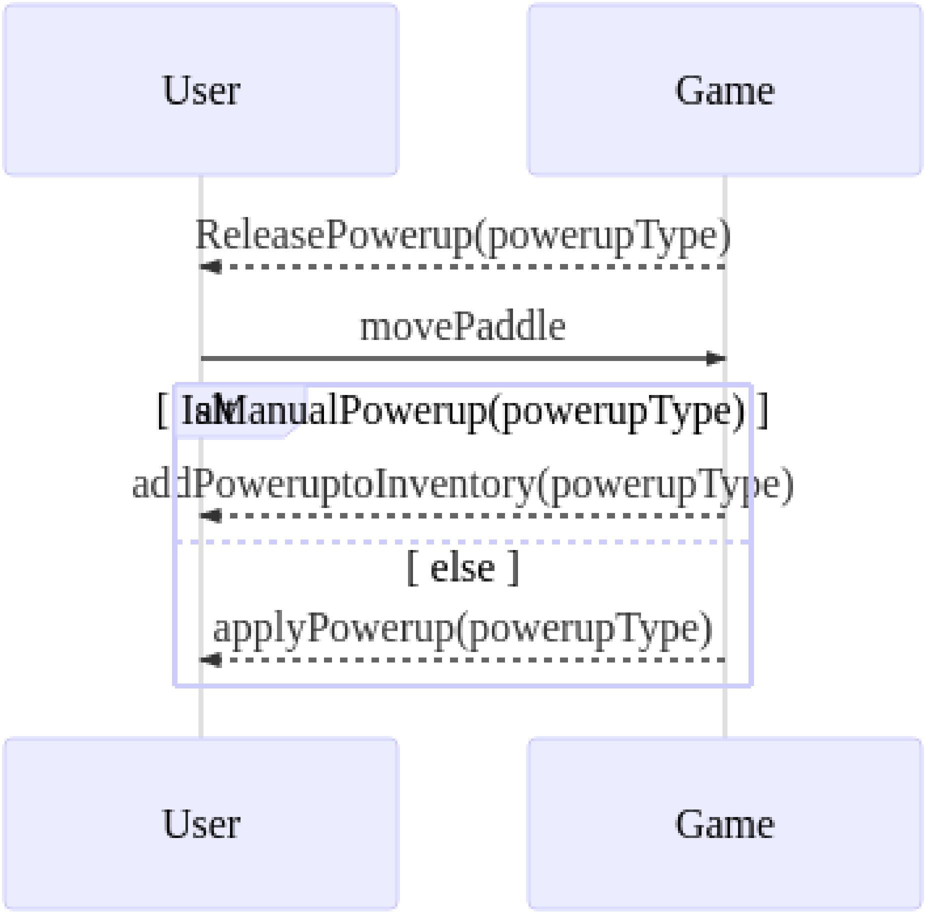
Authenticate Game Sequence Diagram


User
Game
authenticate(username, password)
alt
[isValid?(username,password)]
showBuildMode
[else]
showInvalidCredMsg
User
Game

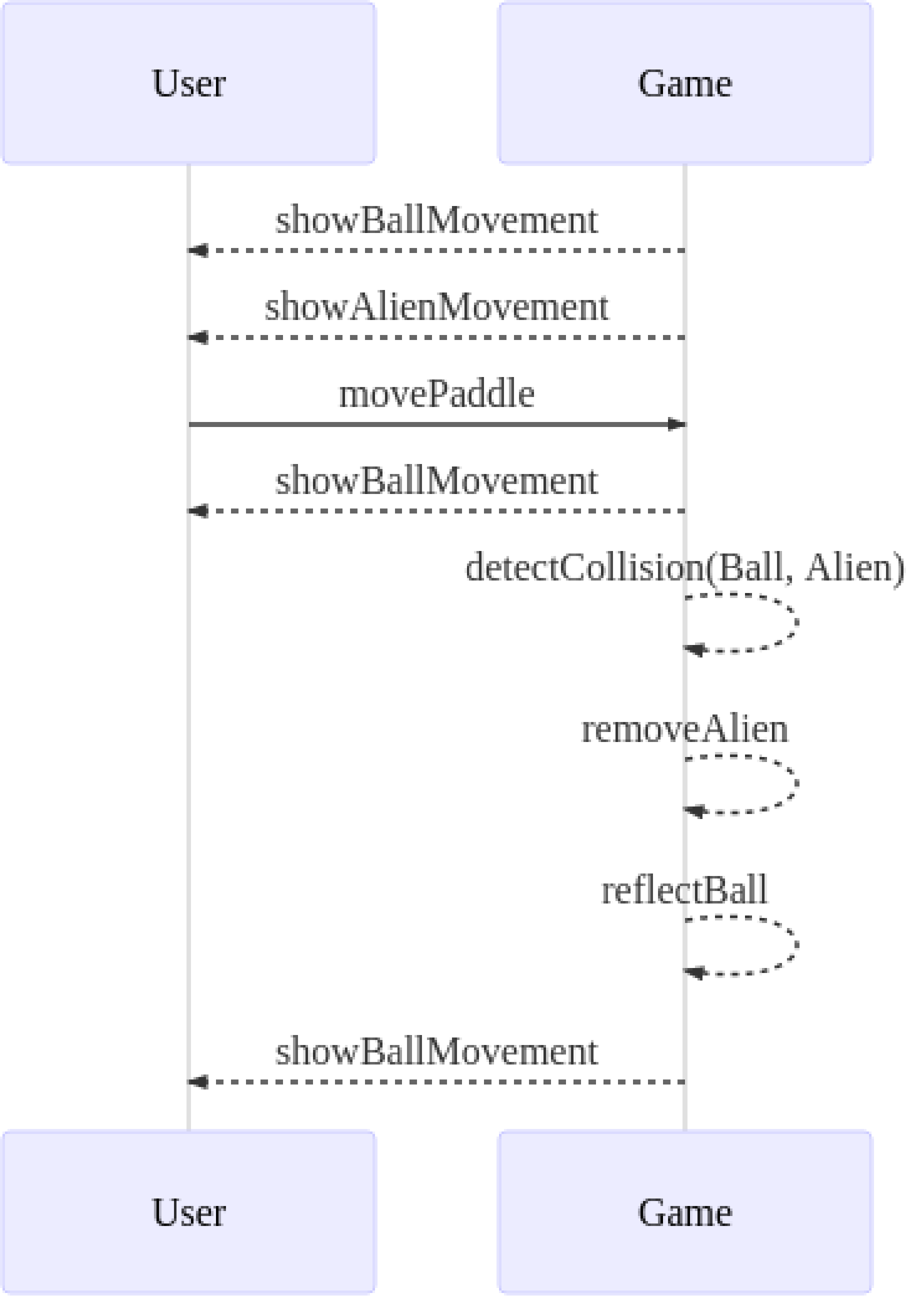
Build a Map Game Sequence Diagram



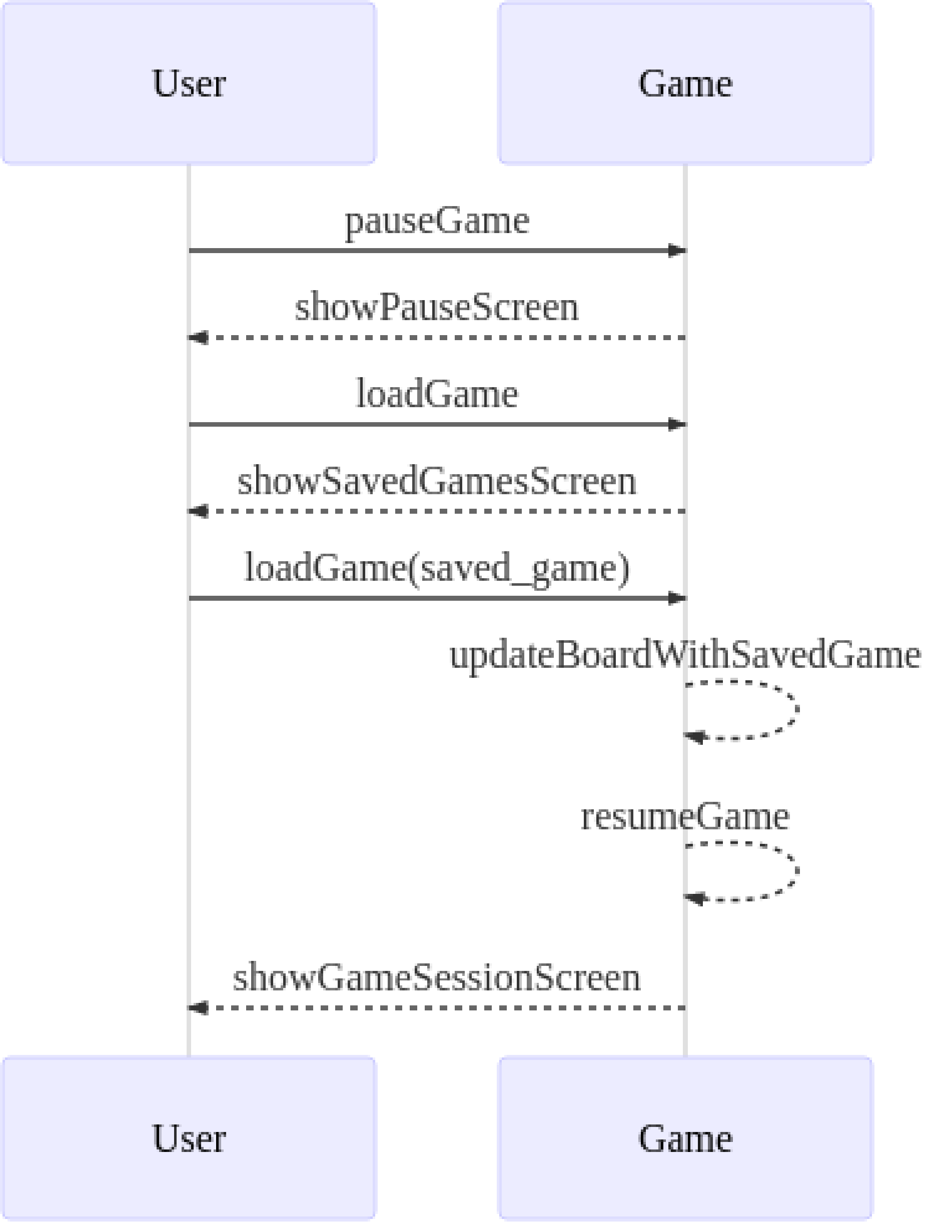
Get Power-Up Game Sequence Diagram


User
Game
ReleasePowerup(powerupType)
movePaddle
alt
[IsManualPowerup(powerupType)]
addPoweruptoInventory(powerupType)
[else]
applyPowerup(powerupType)
User
Game

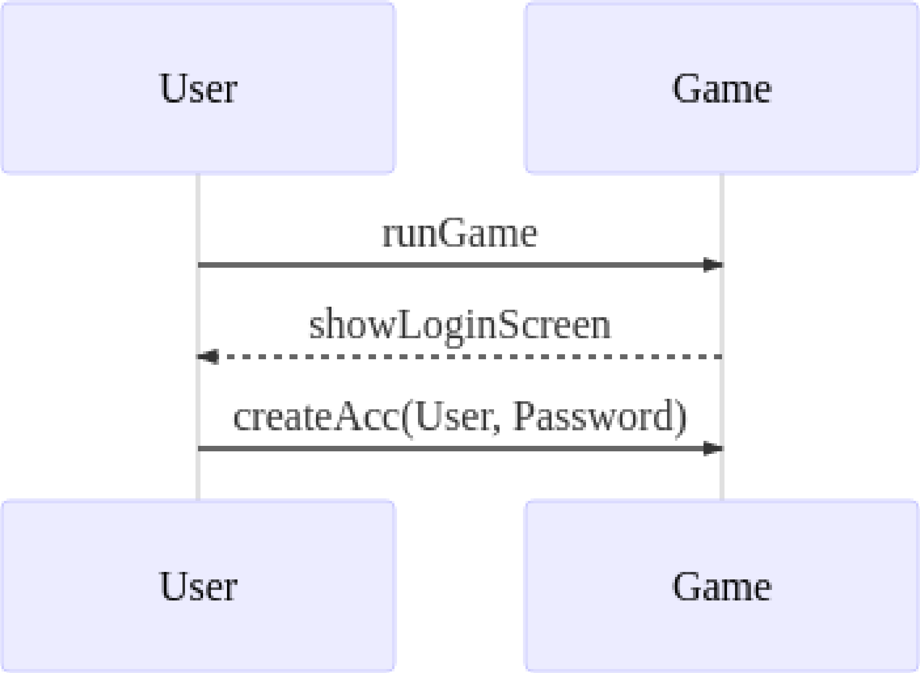
Hit Harmful Alien Game Sequence Diagram



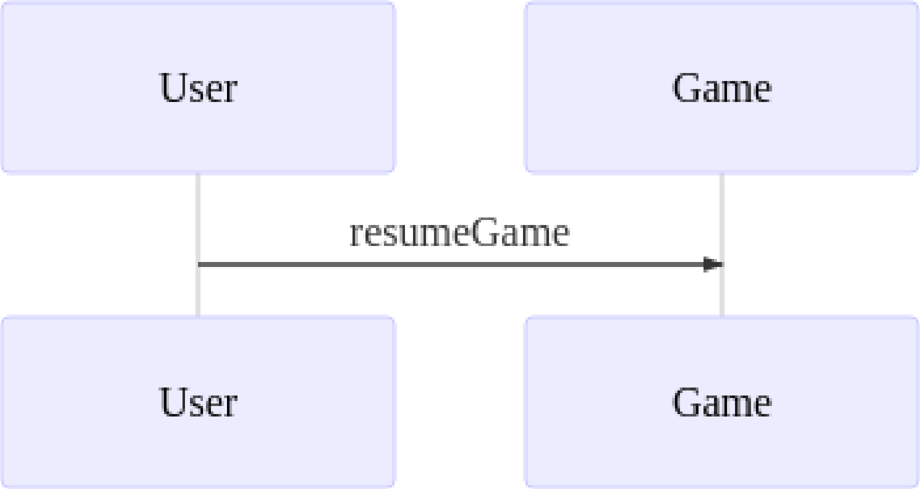
Load a saved game Game Sequence Diagram



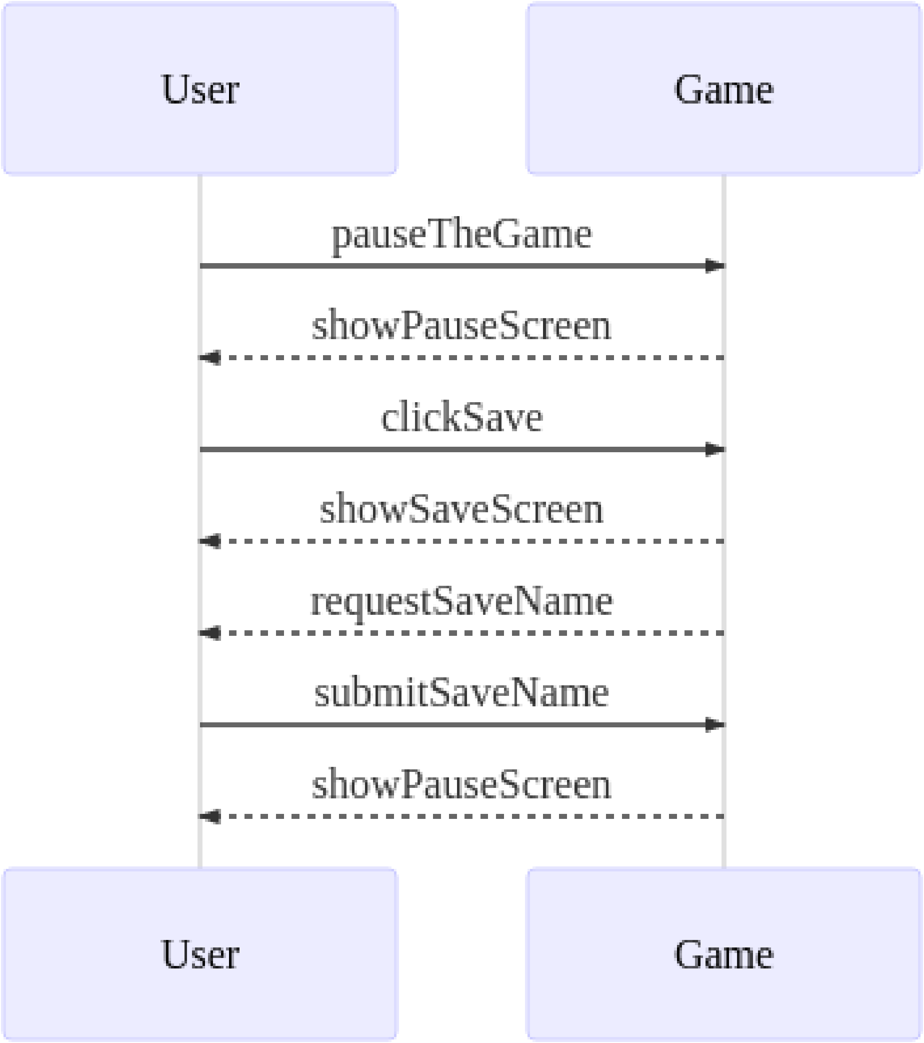
Make an Account Scenario Sequence Diagram



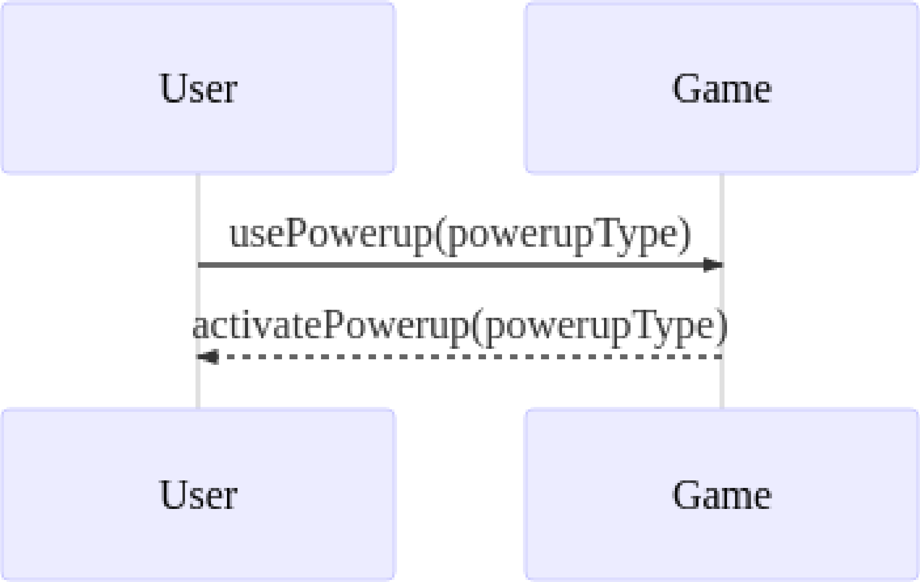
Resume the game Game Sequence Diagram



Save the game Game Sequence Diagram



Use power up Game Sequence Diagram



Operation contracts

Contract CO1: requestHelpScreen

Operation: requestHelpPage()

Cross references: Use Cases: access-help-screen

Preconditions: The Game is paused or is on the login screen

Postconditions:

- Access help screen was displayed

Contract CO2: activateBuildingMode

Operation: activateBuildingMode

Cross references: Use Cases: Build a Map

Preconditions: The user is logged in

Postconditions: * The display switched to the building mode screen * The cursor of the user was placed in the simple brick field by default

---

Contract CO3: moveBrickToPosition

Operation: moveBrickToPosition(brick : Brick, position : Position)

Cross references: Use Cases: Build a Map

Preconditions: The user is on the building mode screen and there is at least one brick in the map

Postconditions: * The brick was placed in the given position * Brick object *brick* was gotten * brick.position becomes position

---

Contract CO4: saveMap

Operation: saveMap(mapName : String)

Cross references: Use Cases: Build a Map

Preconditions: The user is on the building mode screen

Postconditions: * map.name became mapName * The map was created

---

Contract CO5: setBrickNumbers

Operation: setBrickNumbers(simple: integer, halfMetal: integer, mine: integer, wrapper: integer)

Cross references: Use Cases: Build a Map

Preconditions: The user is on the building mode screen

Postconditions: * A Map instance *map* was created * map was associated with the current map * map.numSimpleBrick became simple * map.numHalfMetalBrick became halfMetal * map.numMineBrick became mine * map.numWrapperBrick became wrapper

---

Contract CO6: createAcc

Operation: createAcc(user: Username, password: Password)

Cross references: Use Cases: Create an Account

Preconditions: The user is on the account creation screen

Postconditions: * An *Account* instance was created * Account.user became Username * Account.password became Password

---

Contract CO7: createNewAcc

Operation: createNewAcc

Cross references: Use Cases: Create an Account

Preconditions: The user is on the login screen

Postconditions: * The Board displayed the account creation screen

---

Contract CO8: showAccountCreationSuccess

Operation: showAccountCreationSuccess

Cross references: Use Cases: Create an Account

Preconditions: The user creates an account

Postconditions: * The Board displayed account creation was successful

---

Contract CO9: hideAlien

Operation: hideAlien()

Cross references: Use Cases: Hit-Harmful-Alien

Preconditions: Game is not paused and at least a harmful alien is on the board.

Postconditions: * Harmful alien instance was removed.

Contract CO10: movePaddle

Operation: movePaddle()

Cross references: Use Cases: Hit-Harmful-Alien

Preconditions: Game is not paused

Postconditions: * Paddle.position was changed.

Contract CO11: showBallMovement

Operation: showBallMovement()

Cross references: Use Cases: Hit-Harmful-Alien

Preconditions: Game is not paused

Postconditions: * Ball.position was shown to user.

Contract CO12: loadGame

Operation: loadGame(saved_game: GameMetaData)

Cross references: Use Cases: load-saved-game

Preconditions: There are already saved games

Postconditions: * The Game loaded the selected saved game into the board

---

Contract CO13: requestSavedGamesList

Operation: requestSavedGamesList()

Cross references: Use Cases: load-saved-game

Preconditions: There are already saved games

Postconditions: * User received a list of saved games

Contract CO14: requestResumeGame

Operation: requestResumeGame()

Cross references: Use Cases: resume-the-game

Preconditions: There are already a paused games

Postconditions: * The Game resumed the paused game and hided the pause screen

---

Contract CO15: enterSavename

Operation: enterSaveName()

Cross references: Use Cases: save-the-game

Preconditions: user clicked save button

Postconditions: * Box appeared to user prompting him to enter a save name.

Contract CO16: printSaveSuccessful

Operation: printSaveSuccessful()

Cross references: Use Cases: save-the-game

Preconditions: user submitted a save name to Game

Postconditions: * Message was shown to user notifying of successful operation.

Contract CO17: showPauseMenu

Operation: showPauseMenu()

Cross references: Use Cases: save-the-game

Preconditions: user chooses pause menu option

Postconditions: * save menu was opened for user

Contract CO18: submitSavename

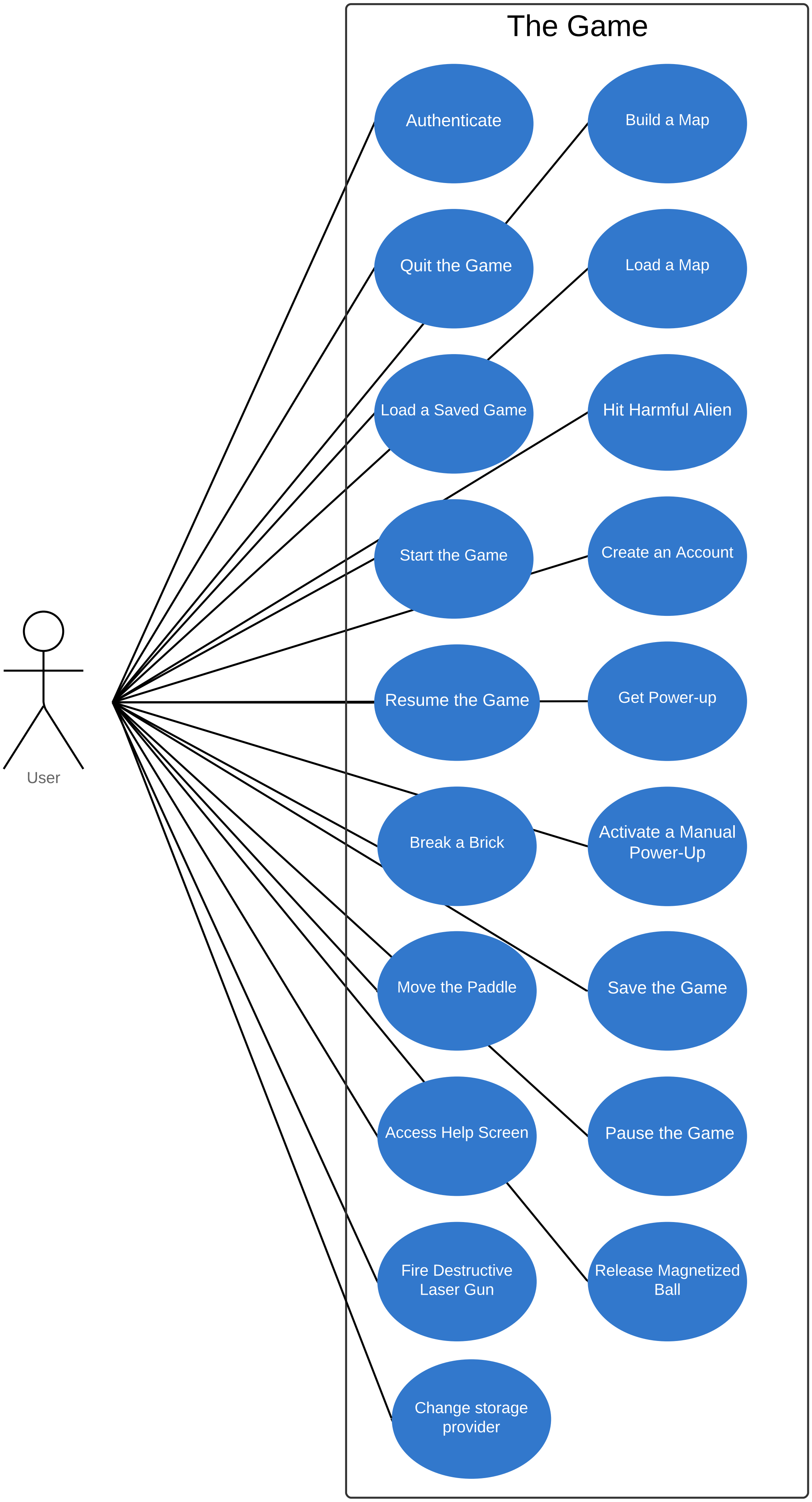
Operation: submitSaveName(saveName)

Cross references: Use Cases: save-the-game

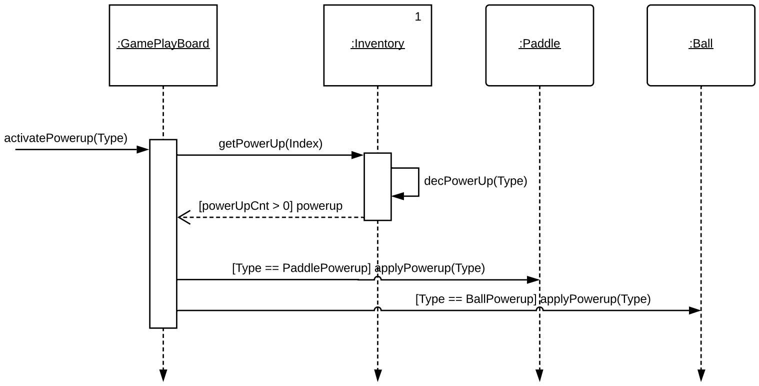
Preconditions: user is prompted to enter a save name

Postconditions: * savename instance was created. * savename instance associated with The Game.

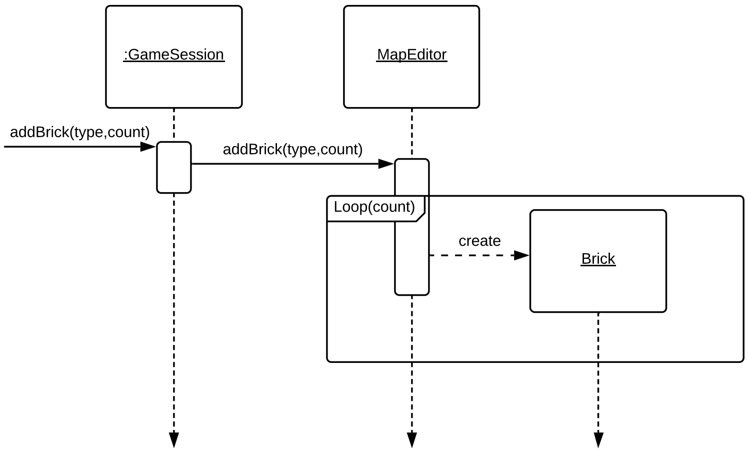
Use Case Diagram



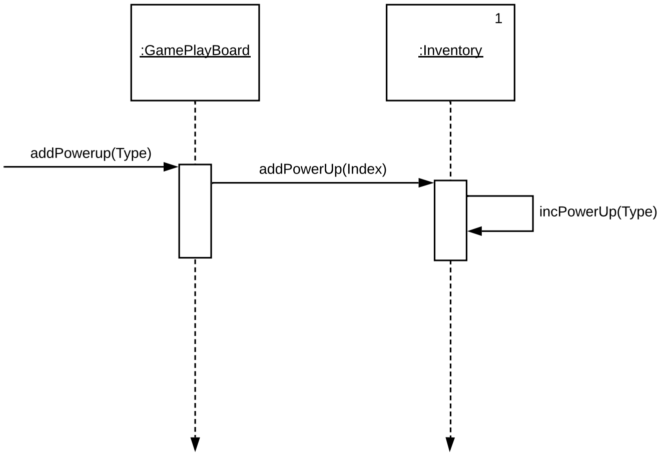
Activate Powerup Sequence Diagram



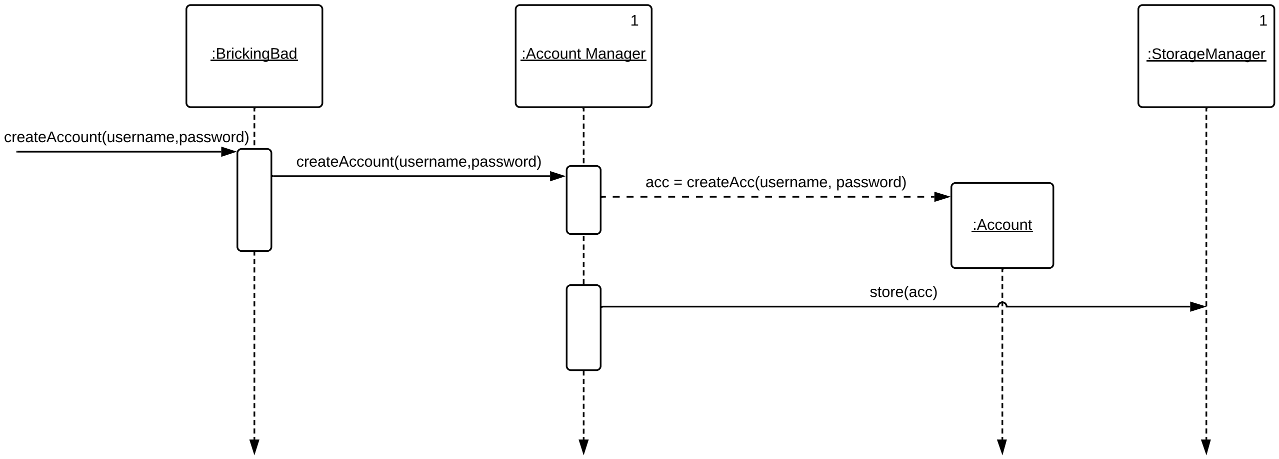
Add Brick Sequence Diagram



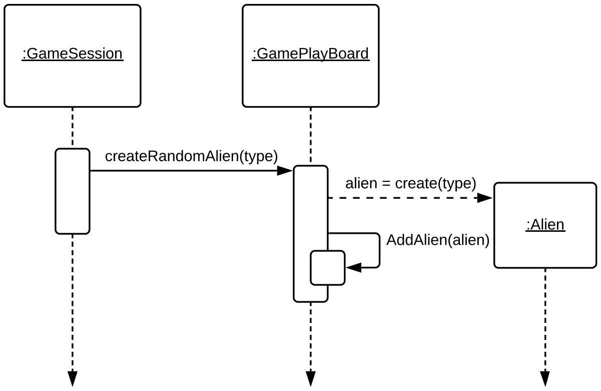
Add Powerup Sequence Diagram



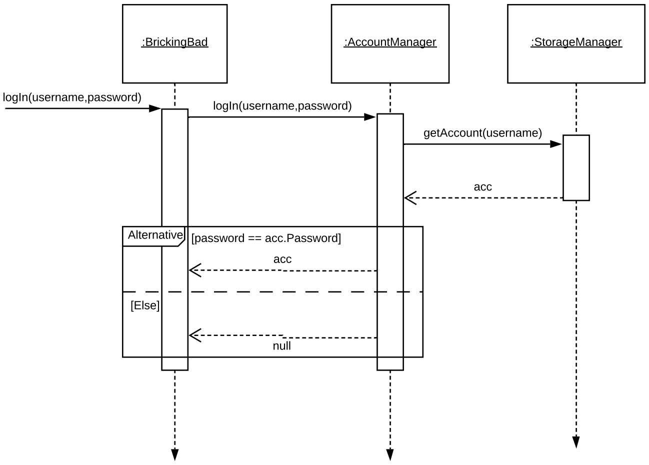
Create Account Sequence Diagram


:BrickingBad
1
:Account Manager
1
:StorageManager
createAccount(username,password)
createAccount(username,password)
acc = createAcc(username, password)
:Account
store(acc)

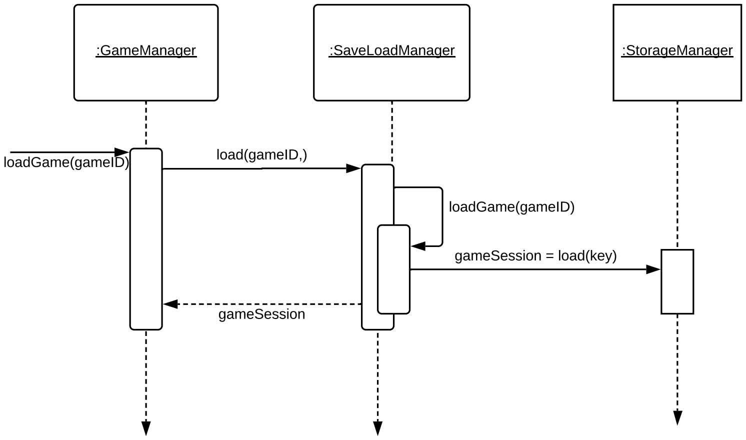
Create Random Alien Sequence Diagram



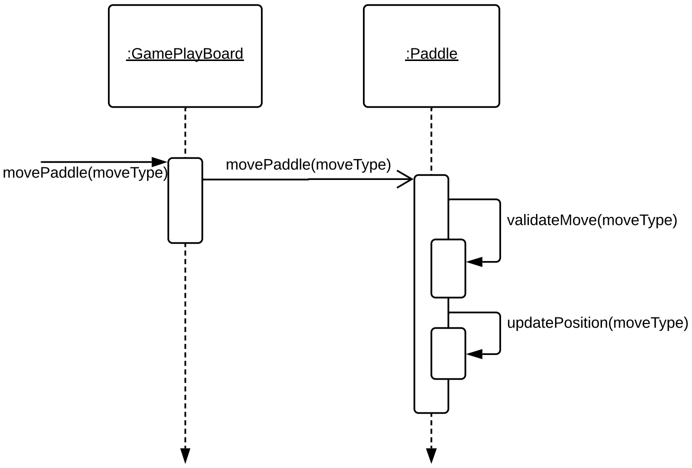
Get Account Sequence Diagram



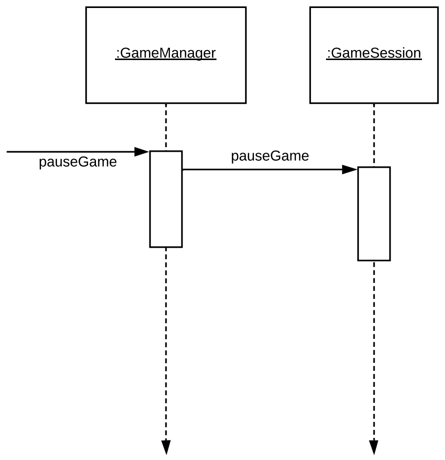
Load Game Sequence Diaram



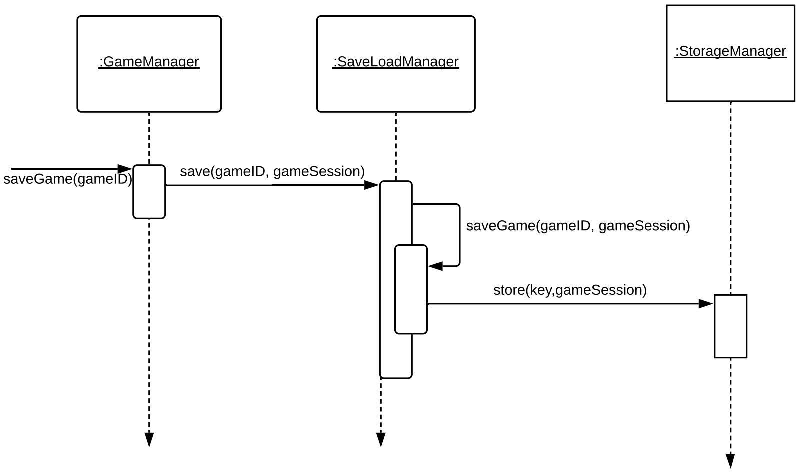
Move Paddle Sequence Diagram



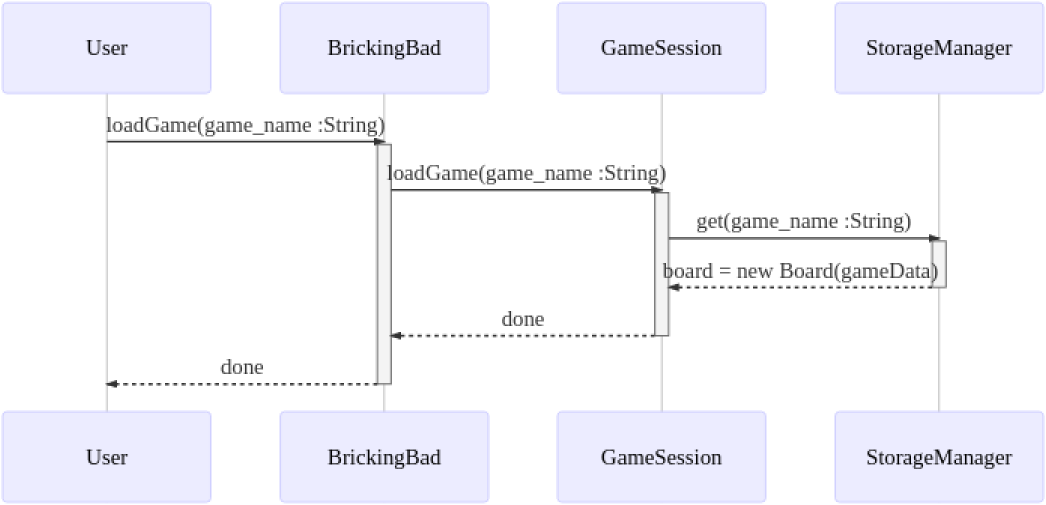
Pause Game Sequence Diagram



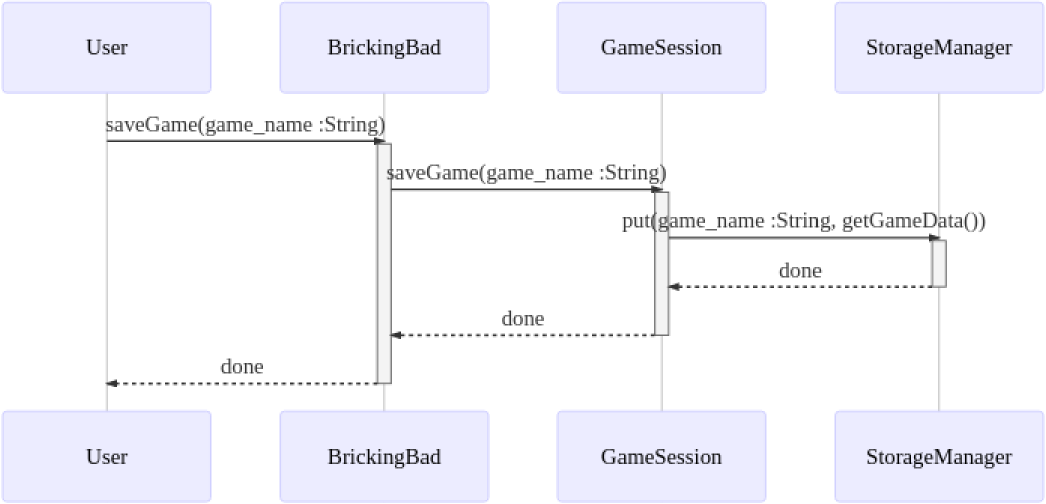
Save Game Sequence Diagram



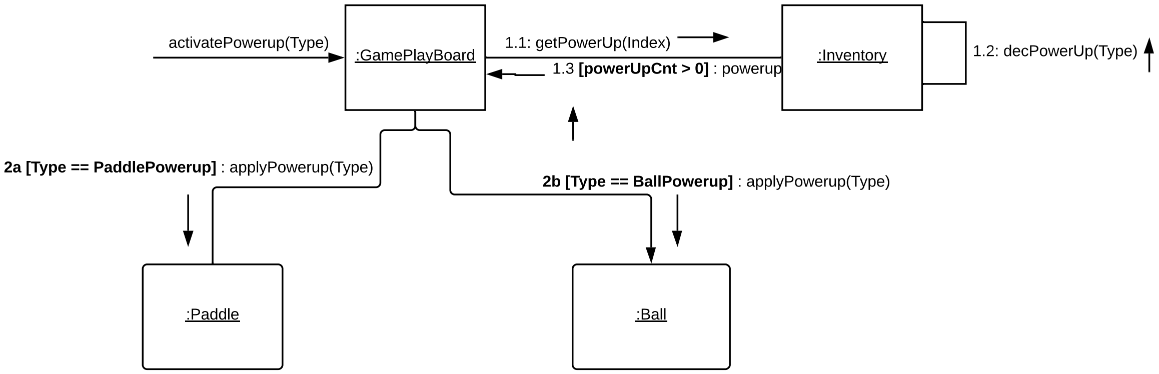
Load a saved game



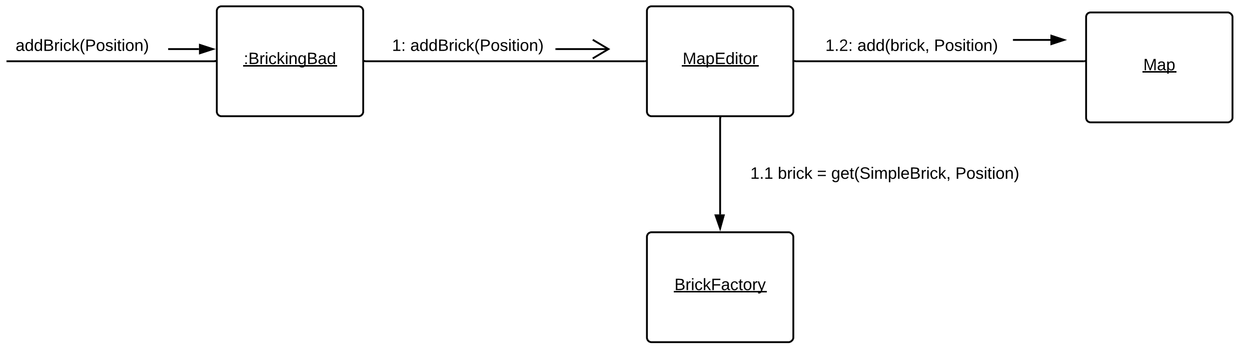
Save current game



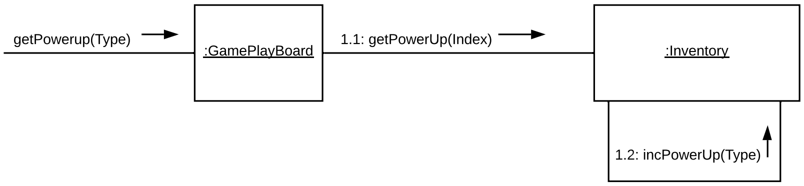
Activate Powerup Communication Diagram


activatePowerup(Type)
:GamePlayBoard
1.1: getPowerUp(Index)
1.3 **[powerUpCnt > 0]** : powerup
:Inventory
1.2: decPowerUp(Type)
2a [Type == PaddlePowerup] : applyPowerup(Type)
2b [Type == BallPowerup] : applyPowerup(Type)
:Paddle
:Ball

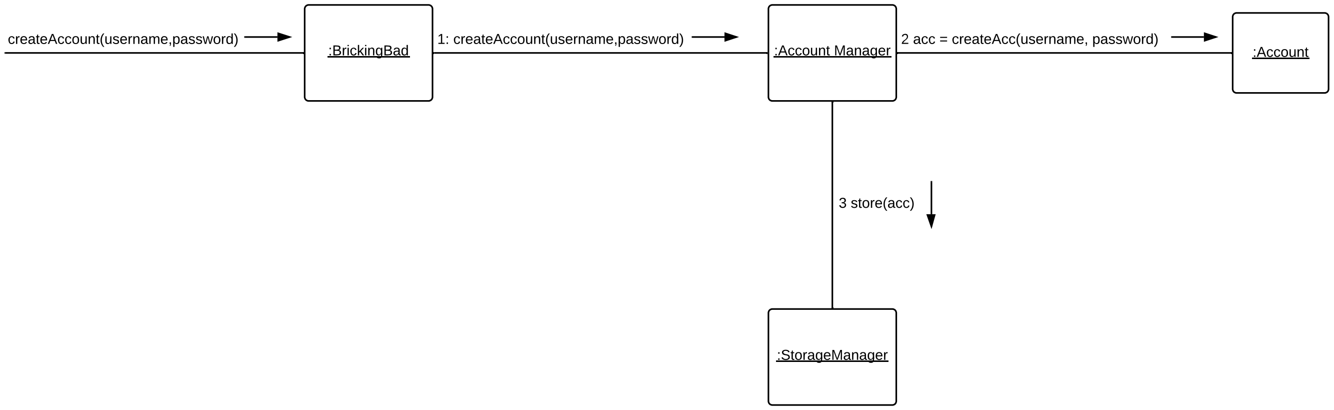
Add Brick Communication Diagram



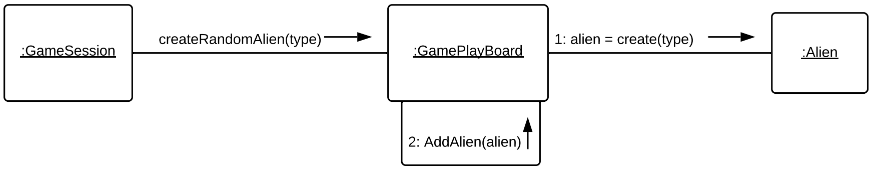
Add Powerup Communication Diagram



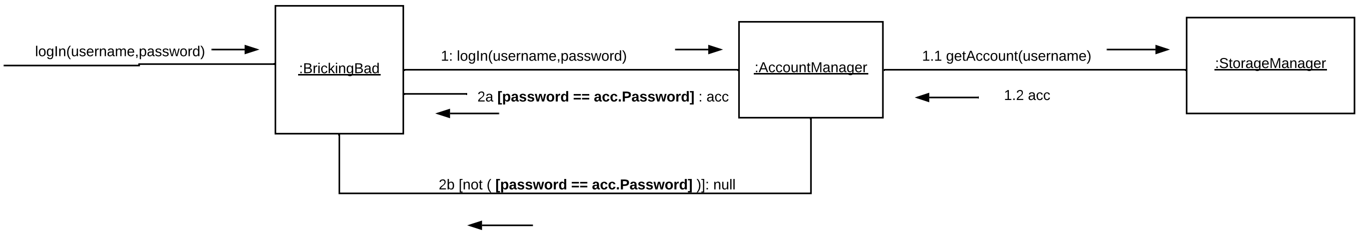
Create Account Communication Diagram


createAccount(username,password)
:BrickingBad
1: createAccount(username,password)
:Account Manager
2 acc = createAcc(username, password)
:Account
3 store(acc)
:StorageManager

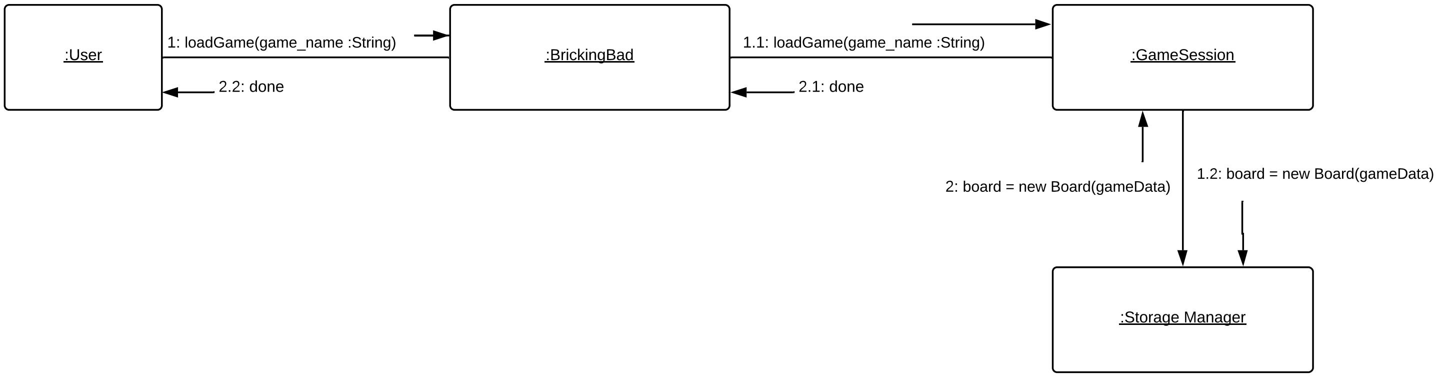
Create Random Alien Communication Diagram



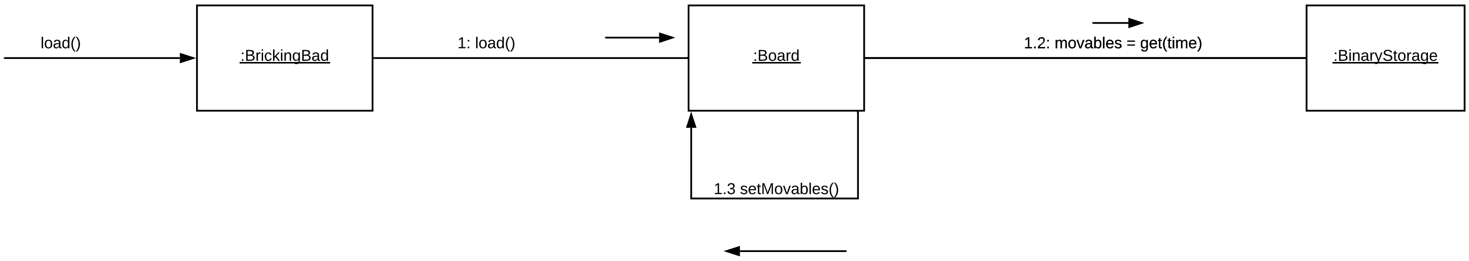
Get Account Communication Diagram


logIn(username,password)
:BrickingBad
1: logIn(username,password)
2a **[password == acc.Password]** : acc
:AccountManager
1.1 getAccount(username)
1.2 acc
:StorageManager
2b [not (**[password == acc.Password]**)]: null

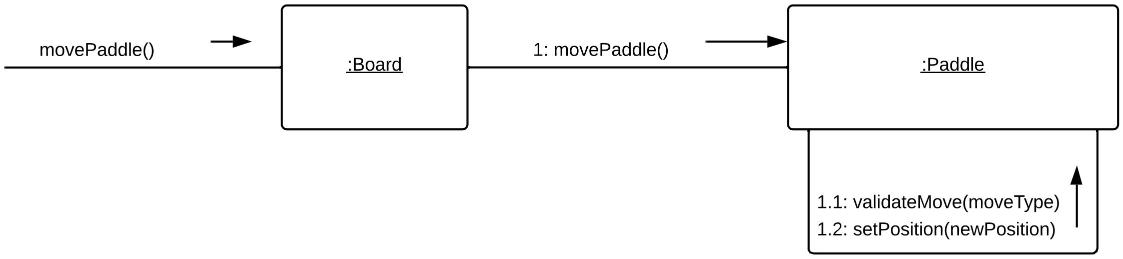
Load a Saved Game Through Controller Communication Diagram


:User
1: loadGame(game_name :String)
2.2: done
:BrickingBad
1.1: loadGame(game_name :String)
2.1: done
:GameSession
2: board = new Board(gameData)
1.2: board = new Board(gameData)
:Storage Manager

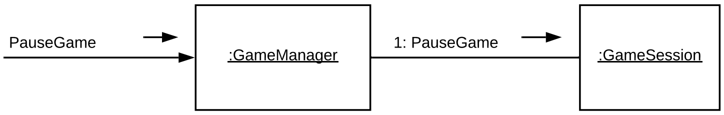
Load Game Communication Diagram



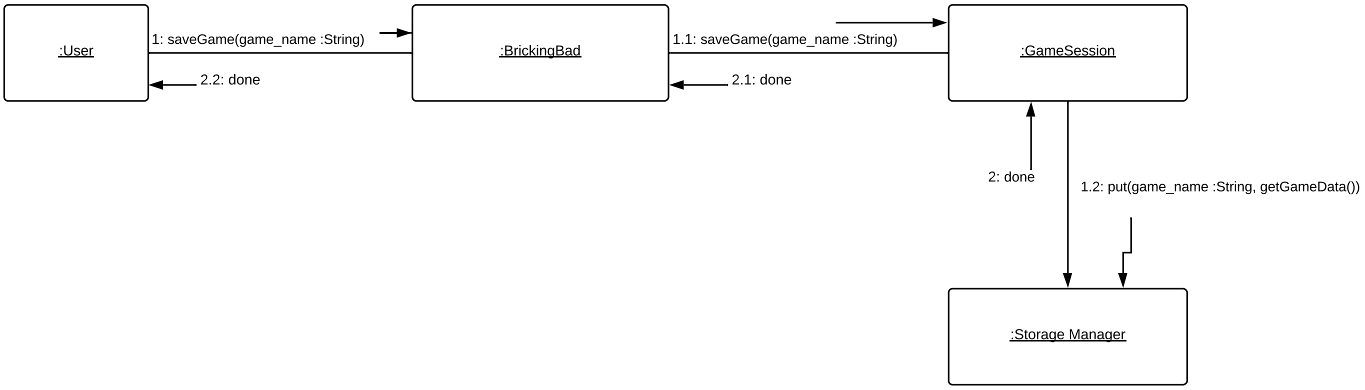
Move Paddle Communication Diagram



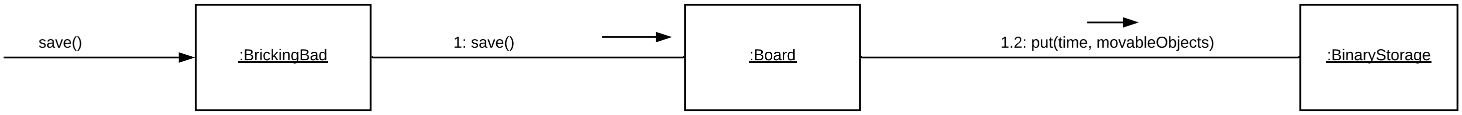
Pause Game Communication Diagram



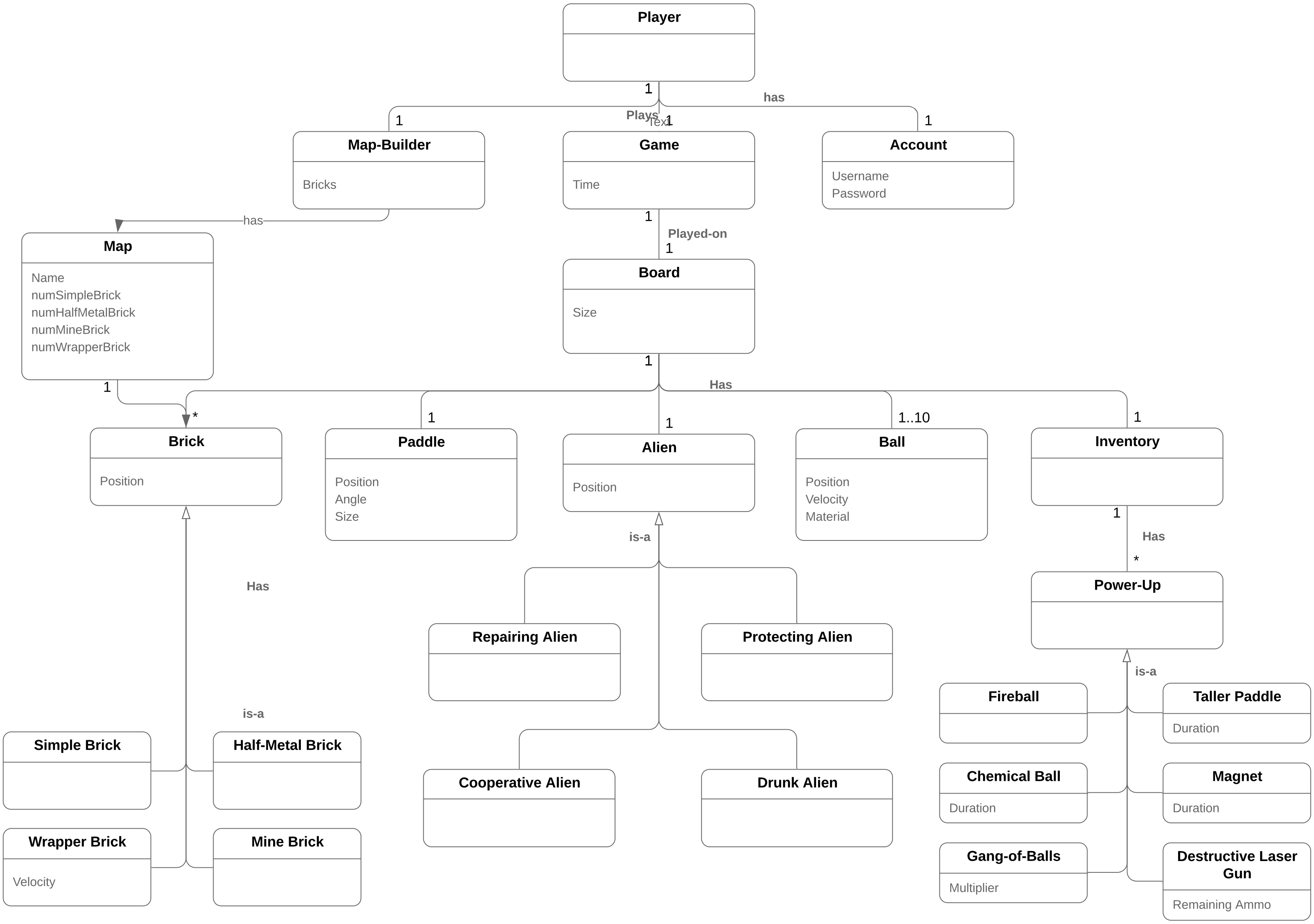
Save Current Game Through Controller
Communication Diagram


:User
1: saveGame(game_name :String)
2.2: done
:BrickingBad
1.1: saveGame(game_name :String)
2.1: done
:GameSession
2: done
1.2: put(game_name :String, getGameData())
:Storage Manager

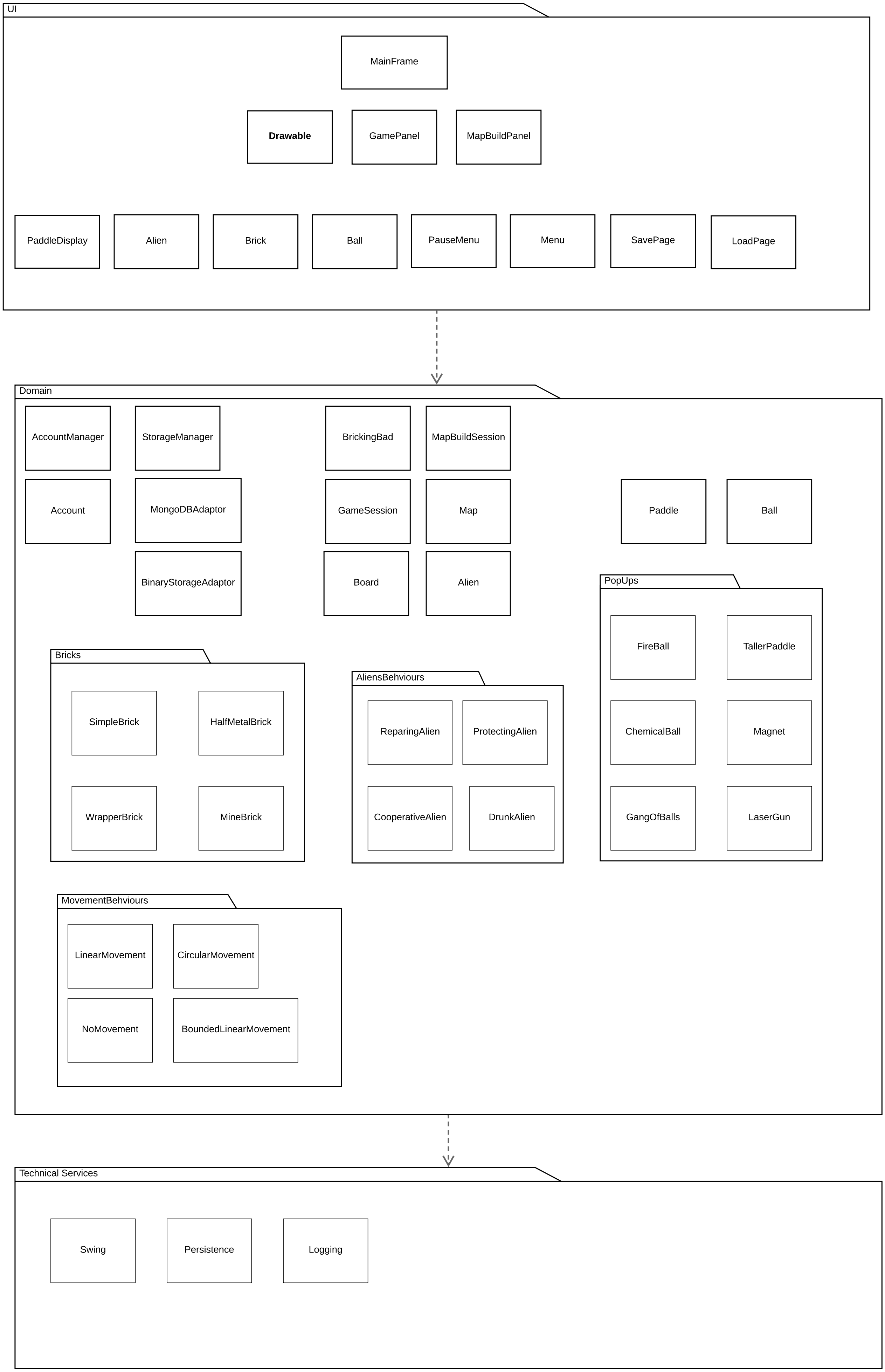
Save Game Communication Diagram



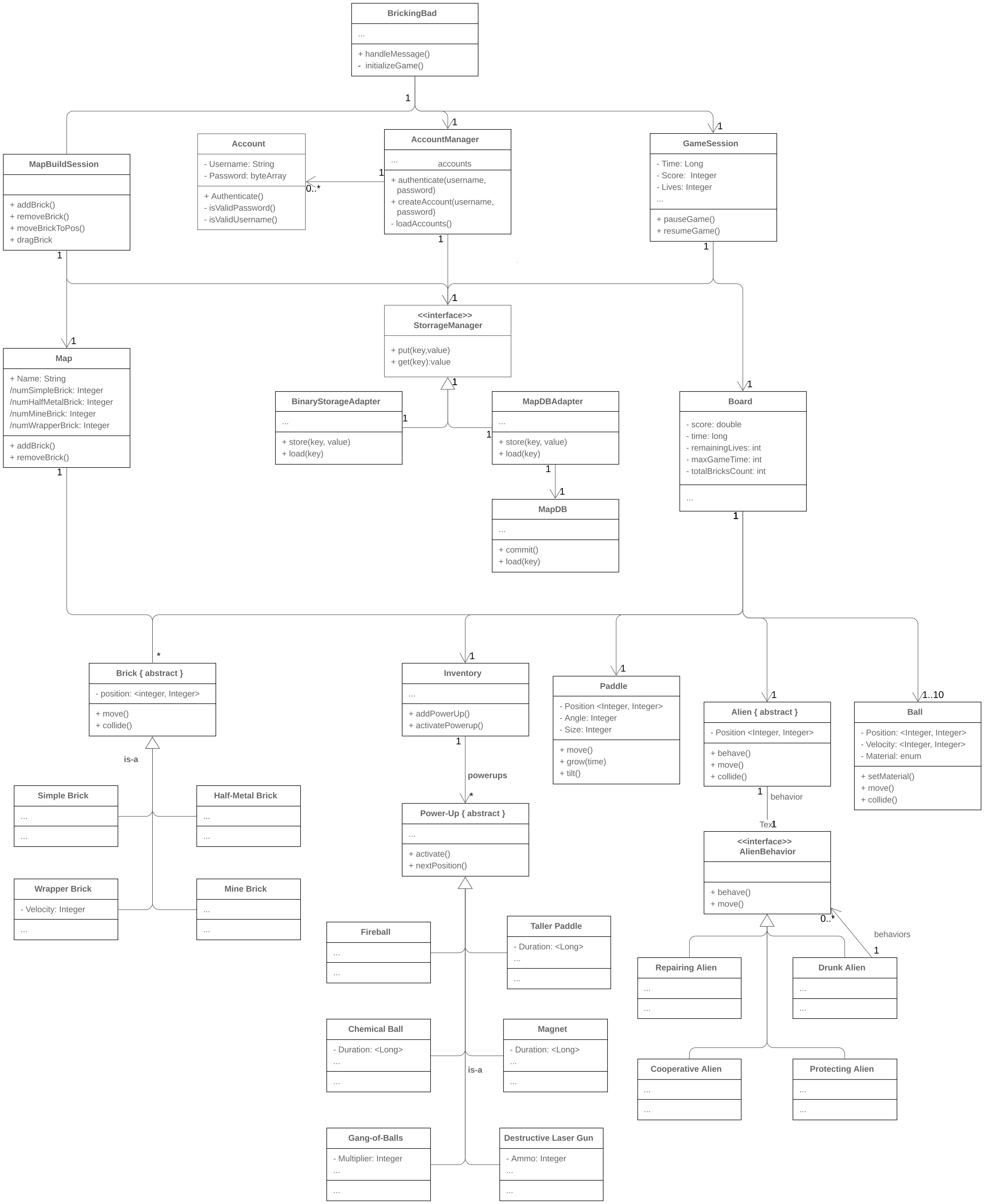
Domain Model



Package Diagram



Class Diagram



Test Plan

PhysicsEngine:

Public Methods: - calculateNewVelocity
- Side hit - Corner hit - calculateCollisionSlope - Side hit - Corner hit - isCollided
- Collided balls - Not collided balls - Collided ball with rectangle - Not collided ball with rectangle - Collided rectangle with rectangle - Not collided rectangle with rectangle - Self collision - relativeXDirection
- Other object to the left - Other object to the right - relativeYDirection
- Other object to the top - Other object to the bottom - CalculatePostCollisionVelocity - Horizontal wall - Vertical wall - Slanted walls: - Y = -x - Y = x

Account Manager Public Methods: - Register - Register a user that does not exist in the account manager - Authenticate
- Authenticate a user that exists in the account manager - Fail to Authenticate a user that does not exist in the account manager

Storage Manager

Public Methods: - put - put throws illegalArgumentException when provided with null key or value
- put inserts data correctly to the storage - get
- get throws illegalArgumentException when provided with null key - get returns the correct data for the given keys if it exists - contains
- contains throws illegalArgumentException when provided with null key - contains returns true for data that exists inside the storage - contains returns false for data that does NOT exist inside the storage - Constructor
- Constructor works properly when given proper name - Constructor throws illegalArgumentException when provided with null name

Board

Public Methods:
- Board(GameData)
- Constructor works properly when given proper GameData - Constructor throws null when provided with null GameData - moveAllMovables
- Objects on board move properly according to their move function - removeDestroyedMovables
- Objects that are marked as destroyed are removed from the board - movePaddleLeft
- Paddle moves to left on function call - movePaddleRight
- Paddle moves to Right on function call - rotatePaddleLeft
- Paddle rotates left on function call - rotatePaddleRight
- Paddle rotates right on function call - getData - Data is properly wrapped inside a GameData instance and returned

MapEditor (MapBuildSession)

Public Methods:

- addBrick
  - Not supported brick returns false
  - Add to negative position should return false
  - Add to appropriate position should return true
- removeBrick
  - Removing existing brick returns true
  - Removing non-existing brick returns false

- moveBrick
  - Moving Brick causing collision should return false
  - Moving Brick to empty place should return true

- getData
  - Data is properly wrapped and returned as GameData instance

Map

Public Methods: - Map
- Map constructor should initialize object container - Add
- Movable shape should be added if there is no collision - Movable shape should not be added if there is no collision - Remove
- Existing movable shape should be removed - Move
- Non-existing movable shape should not change movables container - Movable shape should be moved if there is no collision - Movable shape should not be moved if there is collision - getMovables
- Test get movables - getData
- Test get data

Design pattern discussion

Adapter (storage)

The way our game stores data to disk can be served by multiple third party providers. For example there is a third party library called MapDB which we use to achieve persistent storage. However we also wrote our own persistent storage layer using java's object serialization, which transforms objects to byte arrays that we can easily save and load from disk. In order to keep both options viable while allowing ourselves to introduce other third party storage providers, we chose to use the adapter pattern.

Controller (bricking-bad)

Following the Model View separation principle we tried to follow MVC design (Model View Controller). In which we separated the domain model from GUI code and connect them using a controller. The controllers job is to forward all GUI requests to the corresponding part in domain. For example login attempts will be forwarded to the account manager.

Factory (brick, alien, storage, . . .)

Sometimes the creation logic of objects is not trivial. For example wrapper bricks need to be assigned a random object on creation. Aliens require a specific AlienBehaviour on creation. This required us to create factories for varies types in our game. In which we hide all the complex creation logic in one place.

Strategy (path, alien-behaviour)

Aliens, balls, powerups and the bricks are all movable objects in our game. However not all of them share the same movement. Some turn in circles, others move in straight lines left and rights, others might be moving in a straight line then decide to stop. This required us to think of a way to write our movements which allows us to also change it for some objects during runtime. The strategy pattern came to mind. By writing Movement Behaviours that can be s

Composite (drunk-alien-behaviour)

Most of the aliens in our game have a single behaviour during all of their life cycle. However the drunk alien can go through many shifts during its life cycle. It can act as a repairing alien, then become a protecting alien. Here is where the composite pattern comes to play. We composed multiple strategies into one. In the drunk alien which strategy takes effect "now" depends on a set of rules inside of the composite strategy. On usage of the composite strategy it evaluates the world around it, and accordingly chooses an alien behaviour that matches the state of the world.

Singleton (constants from configuration file)

Constants that are present in the games configuration file (game.properties) should be available to all parts of the game. For example many parts of the domain layer need to know about L (a constant that specifies the length of the paddle, used in the calculation of the speed of many objects, and has many other uses).

Information Expert (GameSession and saved games)

GameSession knows the current user and what is on the board, so it contains a reference to the storage containing all the previously saved games of that user.

Creator (Account Manager creates accounts, knows password and username)

The account manager knows the most about accounts present in the system. It has access to the storage of those accounts, has the power to delete and edit accounts. Therefore according to the creator pattern, we chose that the Account Manager will be the class which is responsible for creation of account objects.

Supplementary Specifications

FURPS+

Functionality:
- The user needs authentication to play the game. - When an error occurs in the system, the error should be logged into a specified text file and the game is ended.

Usability
- The game GUI should be visible from 1 meter.
- Colors of the bricks should be visible by color blind people. - The game should demonstrate sound effects of the actions happenning.

Reliability
- The system should continue to execute user commands even if there is a minor error occurs which does not affect the continuity of the game.

Performance
- The performance of the system should be high enough to be able to process frequent interaction between the system and the user.

Supportability
- The game should work on different platforms which has java installed. - The system should be flexible for different configurations such as time limit, health of the player etc.

Noteworthy Hardware and Interfaces - User presses the first letter of the power-up when they want to activate(T for Taller paddle, M for Magnet, and C for Chemical ball) if they are using the keyboard to activate - User presses W to release a ball captured by the Magnet power-up if they are using the keyboard to release - User presses W to fire a Fire Destructive Laser Gun shot if they are using the keyboard to fire - User presses left arrow or right arrow to move the paddle left or right respectively - User presses A or D to rotate the paddle 45 or 135 degrees respectively

Glossary

Term	Definition
Paddle	The paddle is a rod that appears at the bottom of the screen while a game is in progress. The paddle's purpose is to stop the ball from falling out of the screen as well as to collect power ups.
Ball	The ball is a circular object which moves around on the screen and bounces off other objects. Its function is to break bricks that it bounces off
Save Name	The name that the user enters when saving a game to identify the particular state of the game which he is saving
Board	Contains the currently loaded map and keeps track of the state of the map
System	is the system logic behind the game that interacts with the user
Simple Bricks	Brick that can be broken in one hit
Half-Metal-Brick	Brick with two sides, one which is similar to **Simple Brick**, and other side of metal that can be destroyed by some powerups
Mine-Brick	Brick that is circular and explodes once hit
Wrapper Brick	Brack that is destroyed by one hit, but hides powerups of triggers for aliens.
Reparing Alien	Alien that repairs simple bricks
Protecting Alien	Alien that protects the wall by moving horizontally under the bricks
Cooperative Alien	Alien that helps user by randomly choosing a row and destroying it
Harmful Alien	This includes **Repairing Alien** and **Protecting Alien**
Drunk Alien	Alien that acts as any kind of alien according to remaining bricks
Manual Power-Up	A power-up that user must activate manually after getting it
Automatic Power-Up	A power-up that activates as soon as user gets it

Term	Definition
Game	The scope that refers the time when a user is not actually playing the game but the application is running
Gameplay	The scope that refers the time when a user is actually playing the game

Vision

We are as team Shawarma dedicated to creating epic entertainment experiences of retro games starting with a fun revamp of the old school brick breaker.

Keyboard as the old school controller and the old school display you can clealy see pixels are our lovely friends during this adventure so keep them close.

We value our users as providing them up-to-date features which are not in the base version of the game such as a personal account to save games, new allies and enemies in the gameplay such as aliens. If cracking bricks, exploding mines, and cooperating with aliens are exciting for you, you are in the right place.

So what are you waiting for joining the team. Just send an e-mail to comp302@ku.edu.tr to get consent to join our team.