CSE 344 Software Engineering

Software Design Report

Covid Bird

by

Ann Nedime Neşe Rende
Aleyna Polat
Arda Ayvataş
Burcu Ece Kartal
Can Erdoğan
Temel Metehan Çınar

Yeditepe University

Faculty of Engineering

Department of Computer Engineering

Spring 2021

TABLE OF CONTENTS

1.Introduction

1.1. Purpose

This document aims to describe the structure of the game software system with respect to the requirements mentioned in the analysis report and the features they provided to the game design, as well as the detailed design of the components of the game software system and their dynamic relationships.

1.2. System

The game system is proposed to computer game users of all ages who love to play casual games to have fun.

1.2.1. New System

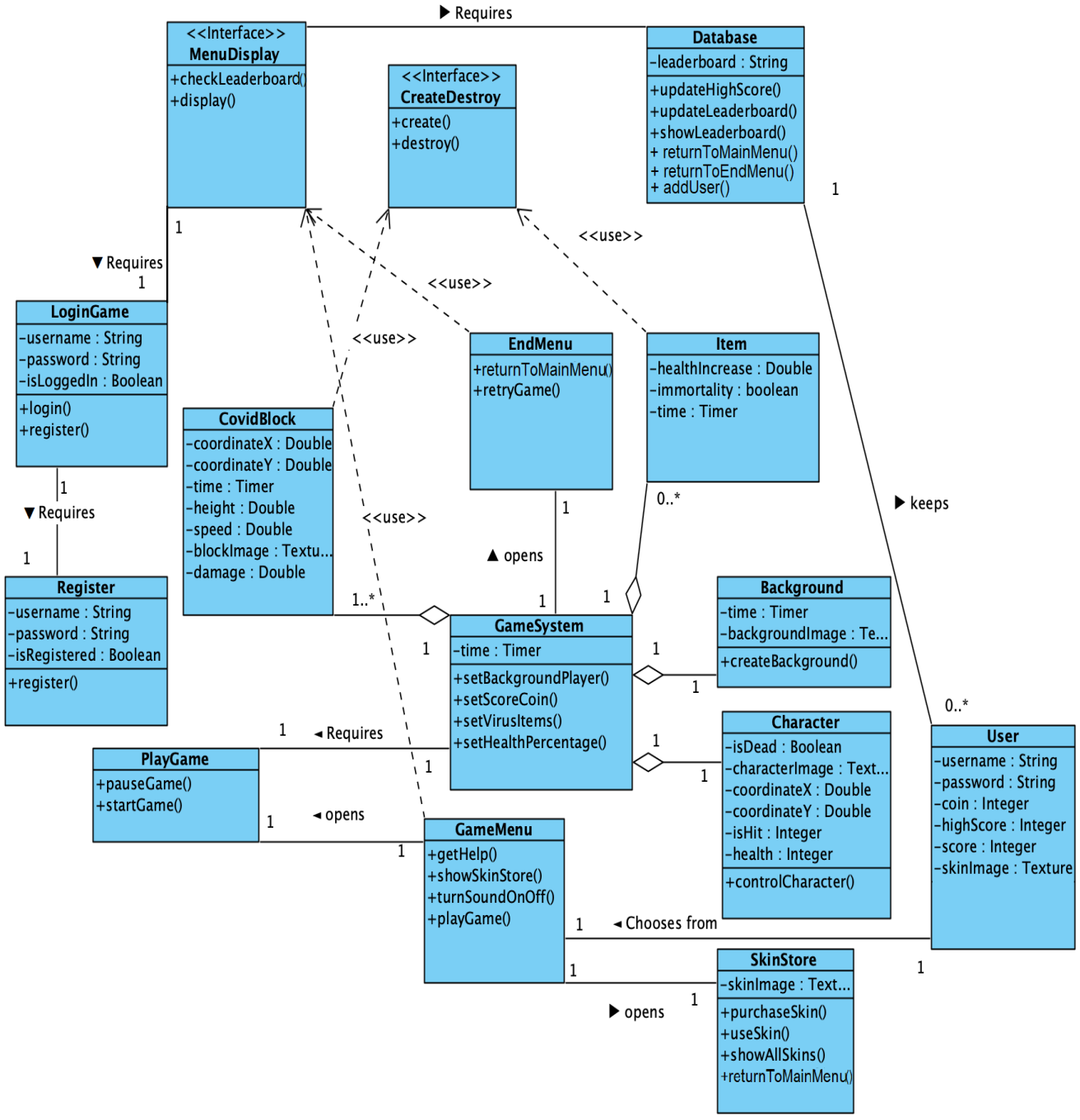
With the addition of a leaderboard, a user can see the top scores all around the world. Users are required to register themselves with their username and password before playing the game. There are several levels of difficulty in the game to make it more competitive. The user can purchase several skins of characters. There is a help option in our game to remind the user of the main rules. New animations and sound effects are added to the game, making it more attractive to players. Our new system also includes the basic features of the games examined within the domain we mentioned in the analysis report.

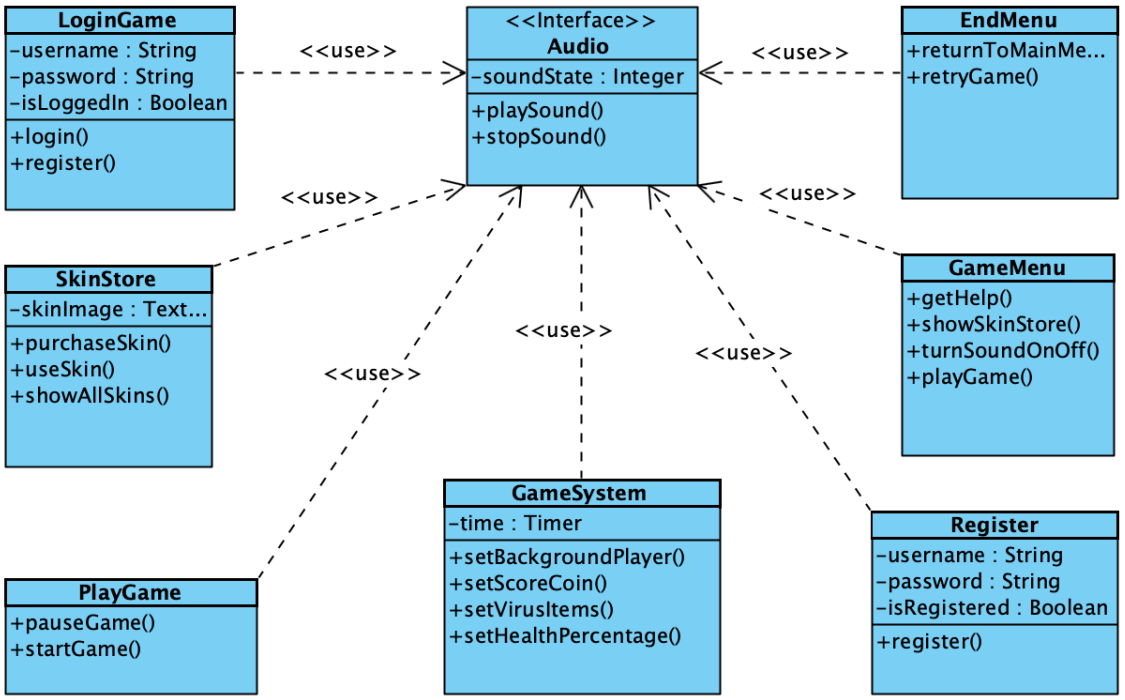
1.3 Structure of the document

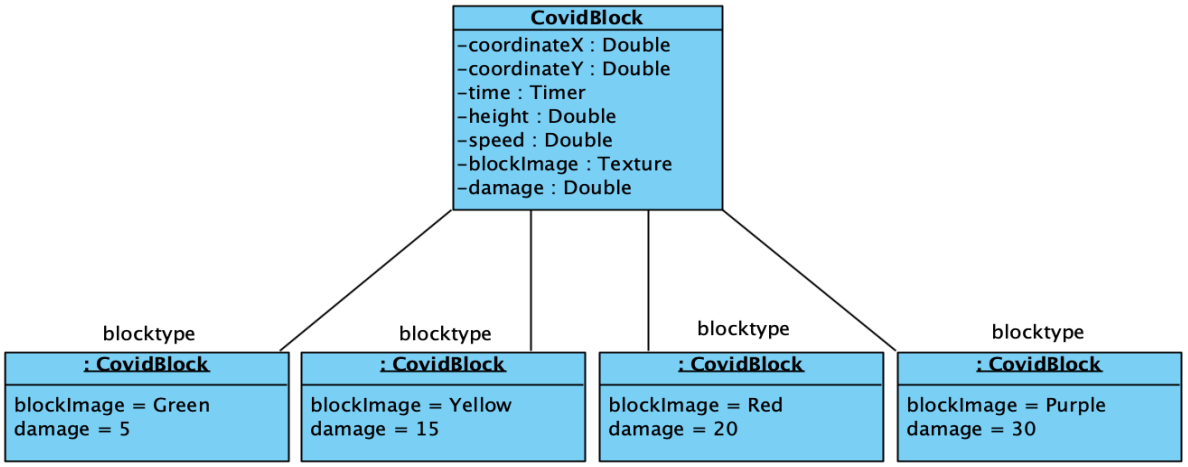
Starting the document with the introduction, we explained what the new system provides to its users. In order to express the system more clearly, we added the detailed design of the class diagram which includes attributes, methods, associations, aggregations, relationship names and multiplicities. With the help of sequence diagrams, we have explained the object interactions in time sequence. After that, we have made a state diagram, so that the behavior of the system can be expressed more clearly. In order to express the concurrent operations and workflows of the activities depending on the user's choices, we included an activity diagram. We used the UML package diagram to simplify our class diagram by grouping the components into packages. After that, we added a component diagram that illustrates how components are bound together to form larger components or software systems. To express the connection between the user, database and skin store, we included an entity-relationship diagram. We finished the document with adding glossary and references.

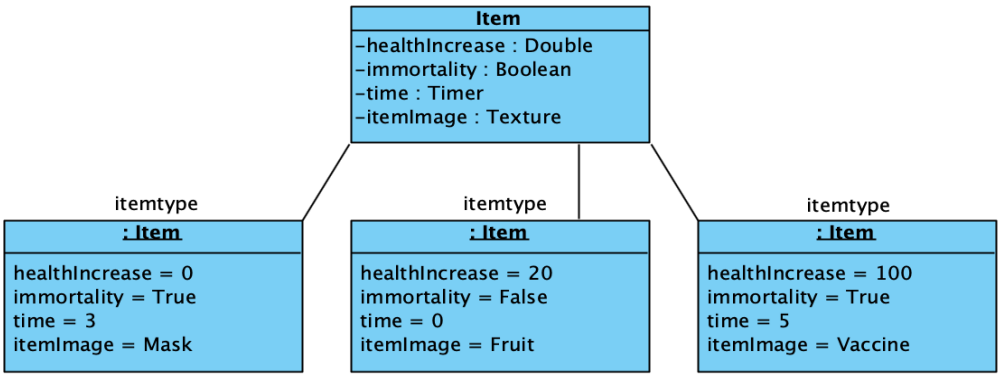
2. Detailed Design of Class Diagram

Associations, aggregations, relationship names and multiplicities used can be seen in the class diagram from the analysis report:

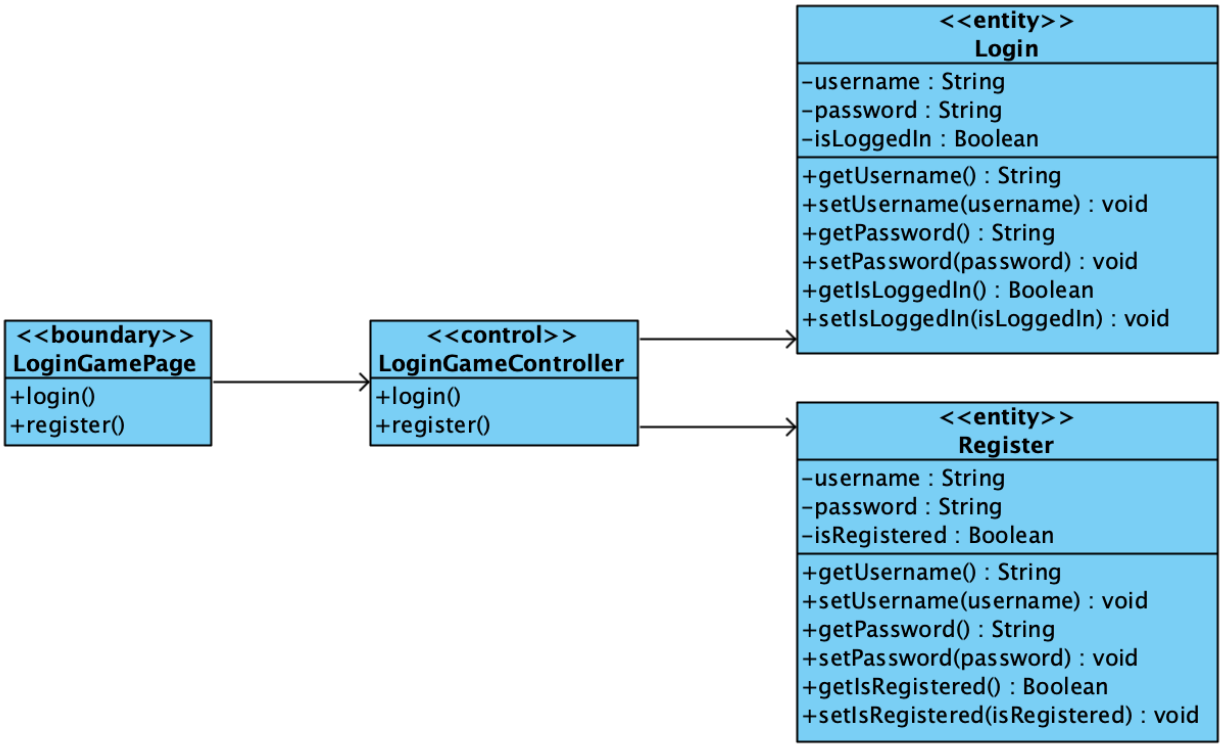


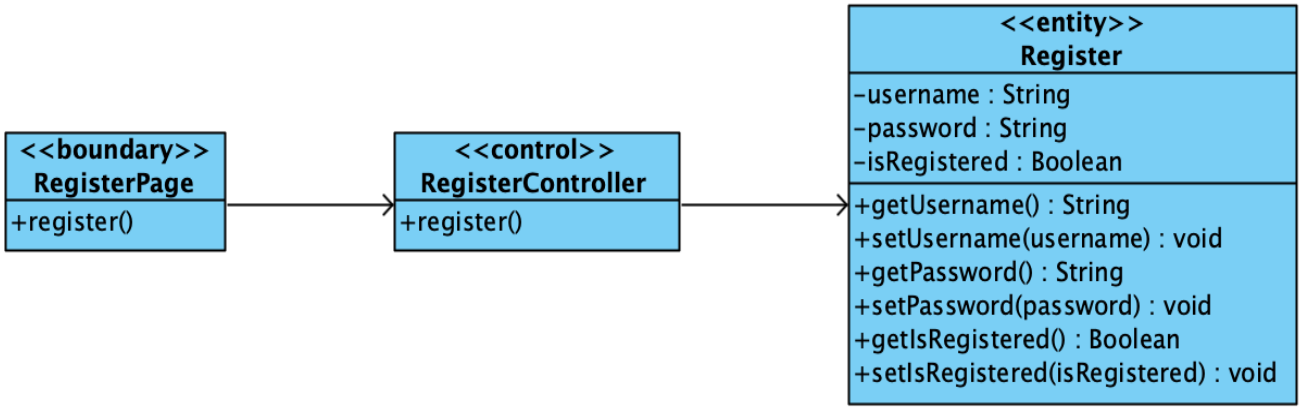

LoginGame
-username : String
-password : String
-isLoggedIn : Boolean
+login()
+register()
<<use>>
<<Interface>>
Audio
-soundState : Integer
+playSound()
+stopSound()
<<use>>
EndMenu
+returnToMainMe...
+retryGame()
<<use>>
<<use>>
<<use>>
SkinStore
-skinImage : Text...
+purchaseSkin()
+useSkin()
+showAllSkins()
<<use>>
<<use>>
<<use>>
GameMenu
+getHelp()
+showSkinStore()
+turnSoundOnOff()
+playGame()
PlayGame
+pauseGame()
+startGame()
GameSystem
-time : Timer
+setBackgroundPlayer()
+setScoreCoin()
+setVirusItems()
+setHealthPercentage()
Register
-username : String
-password : String
-isRegistered : Boolean
+register()

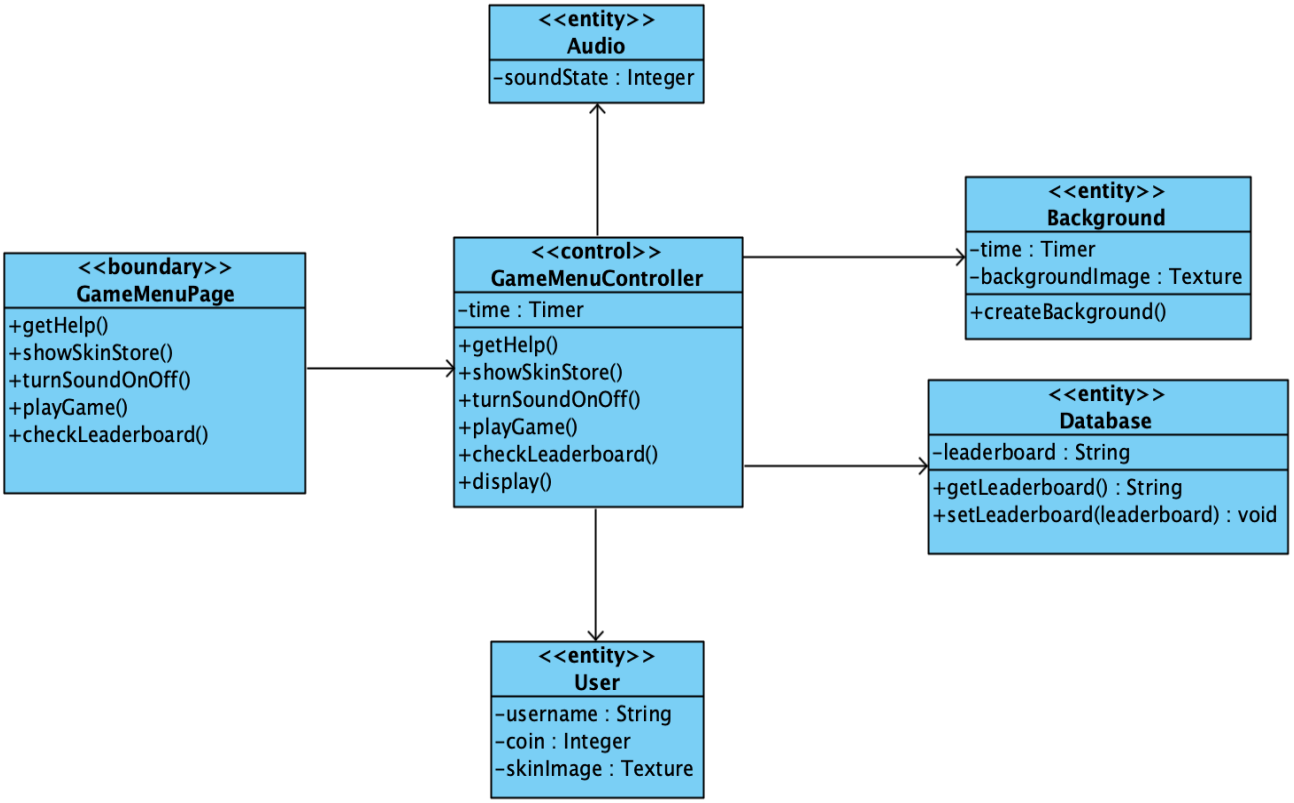

CovidBlock
-coordinateX : Double
-coordinateY : Double
-time : Timer
-height : Double
-speed : Double
-blockImage : Texture
-damage : Double
blocktype
: CovidBlock
blockImage = Green
damage = 5
blocktype
: CovidBlock
blockImage = Yellow
damage = 15
blocktype
: CovidBlock
blockImage = Red
damage = 20
blocktype
: CovidBlock
blockImage = Purple
damage = 30

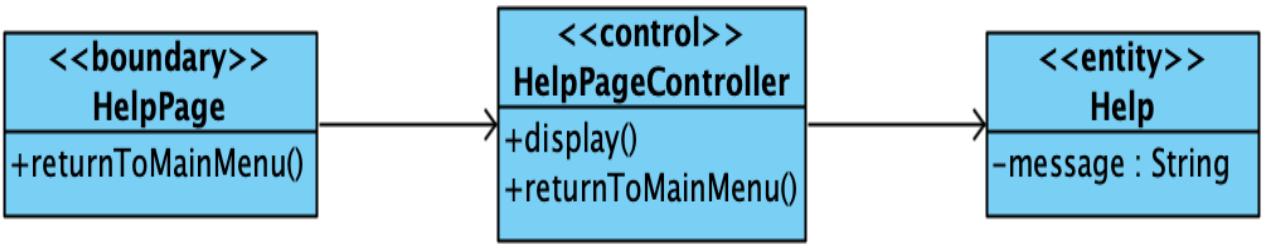

Item
-healthIncrease : Double
-immortality : Boolean
-time : Timer
-itemImage : Texture
itemtype
: Item
healthIncrease = 0
immortality = True
time = 3
itemImage = Mask
itemtype
: Item
healthIncrease = 20
immortality = False
time = 0
itemImage = Fruit
itemtype
: Item
healthIncrease = 100
immortality = True
time = 5
itemImage = Vaccine

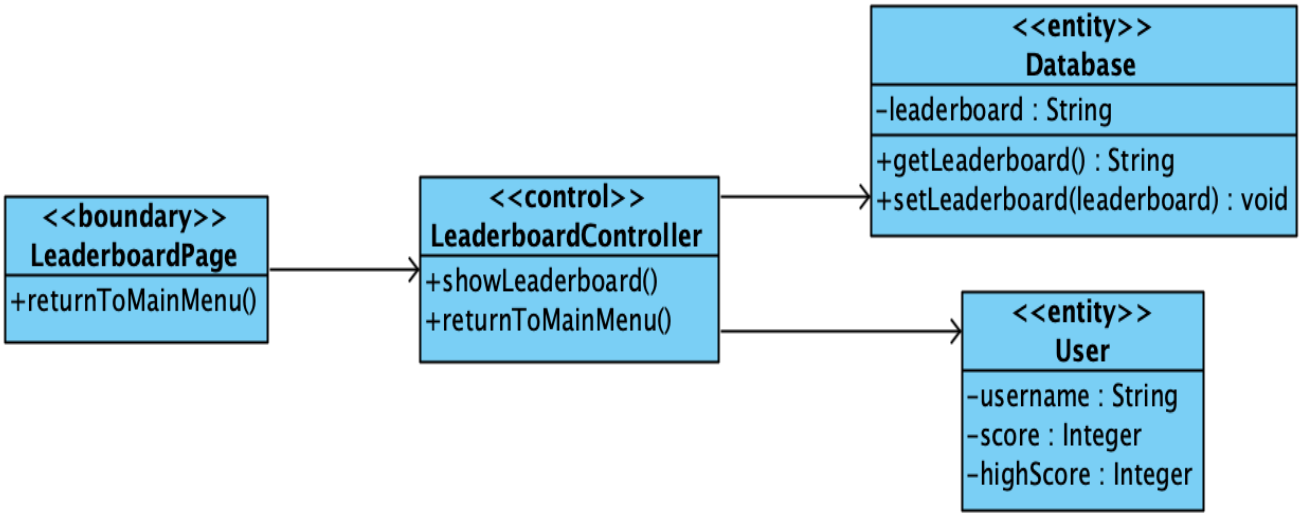
The more detailed form of the parts of the analysis class diagram:

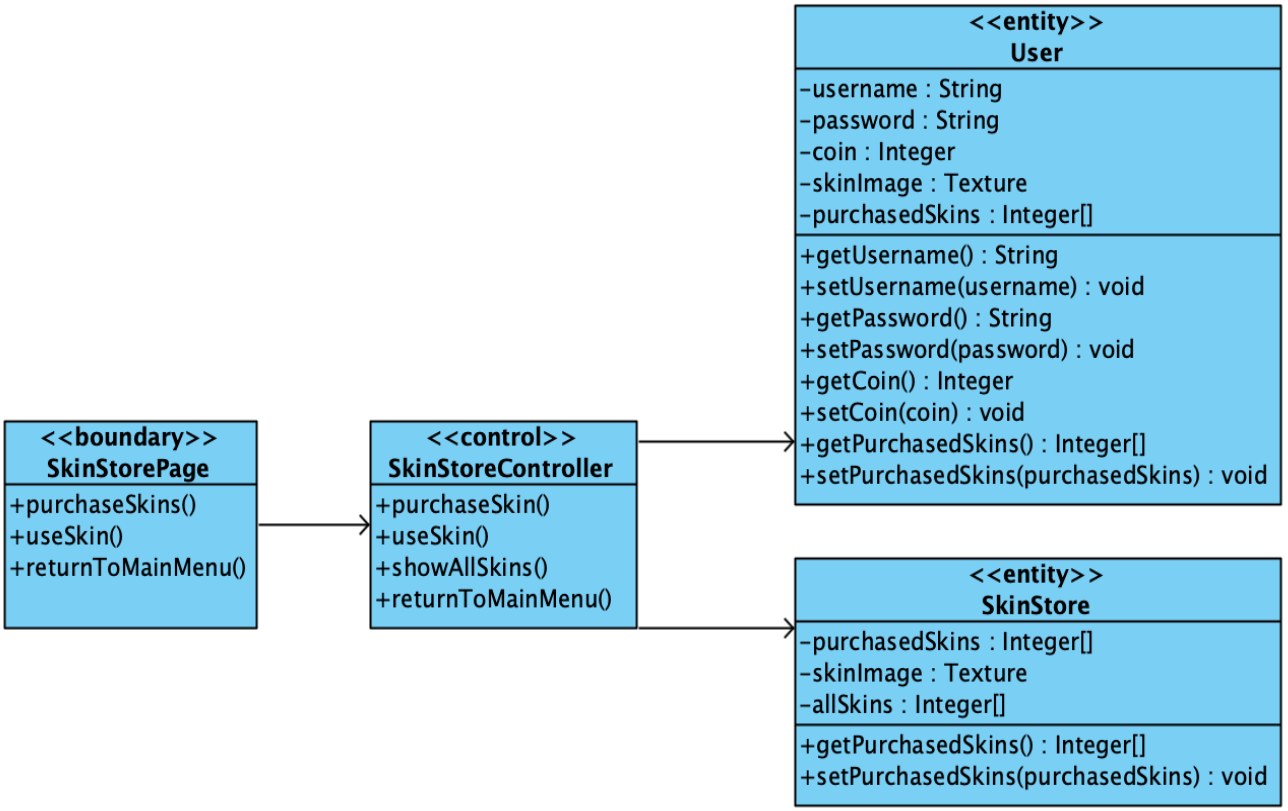


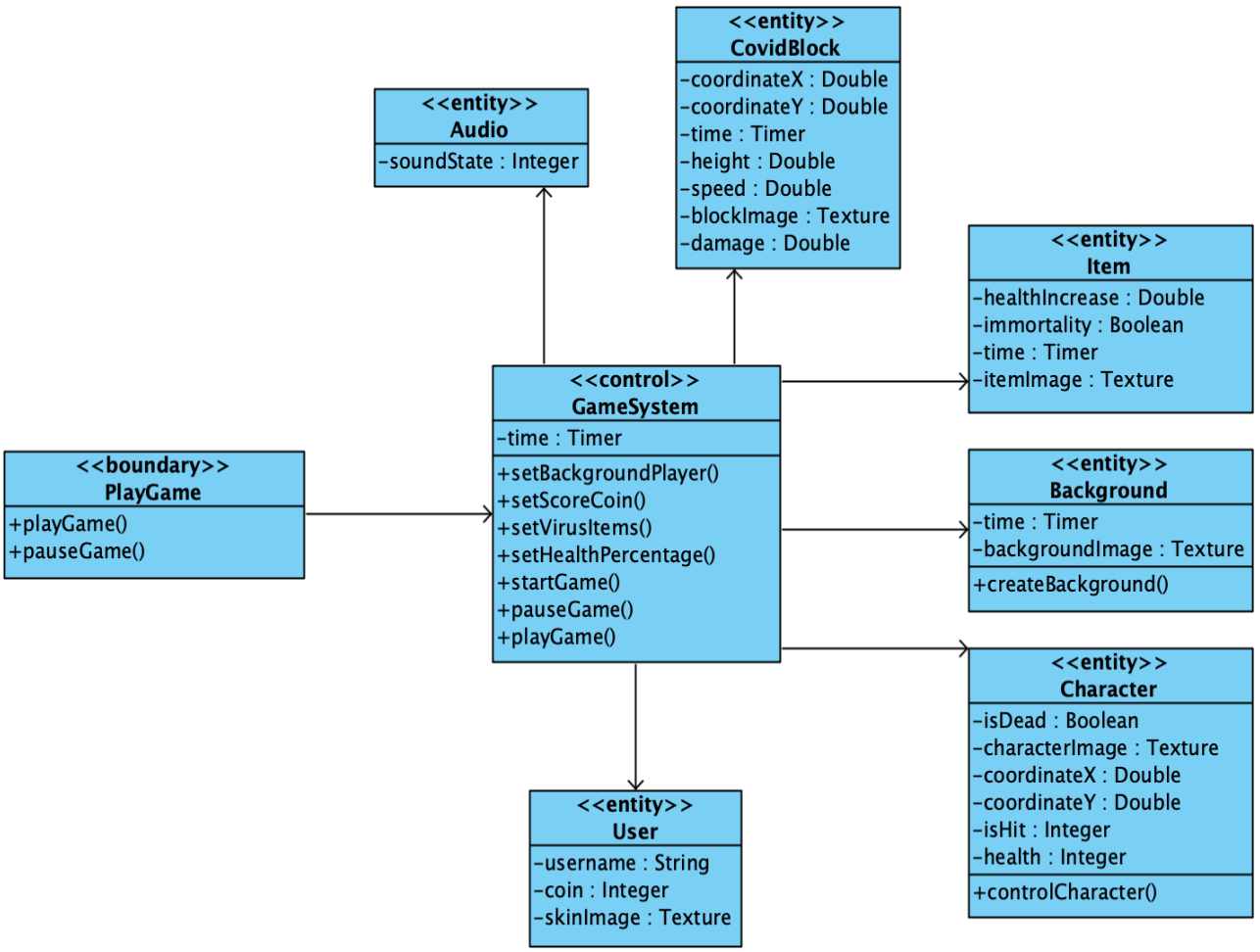


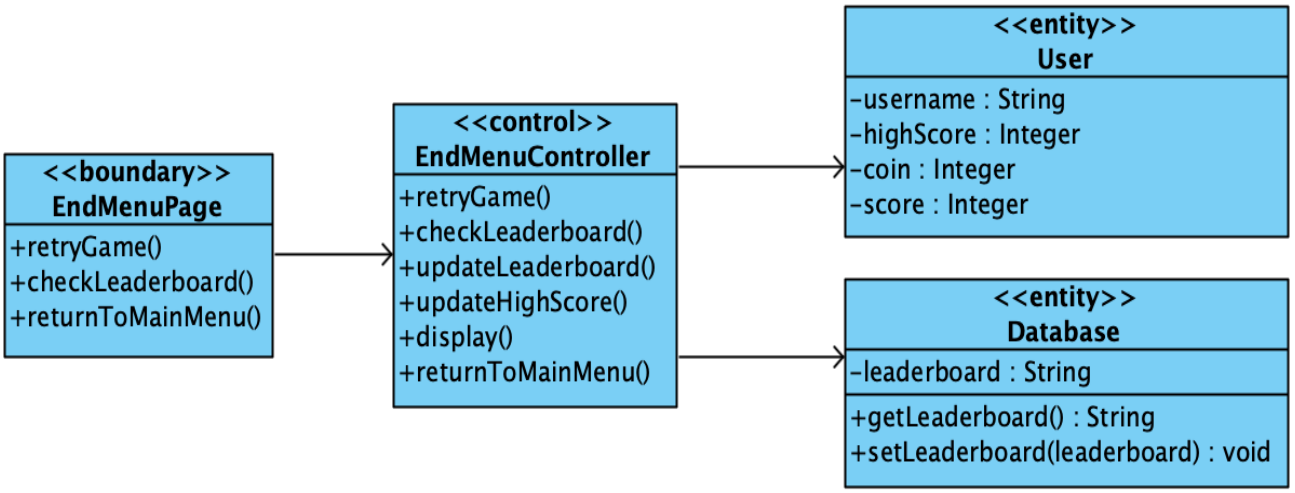

<<entity>>
Audio
-soundState : Integer
<<boundary>>
GameMenuPage
+getHelp()
+showSkinStore()
+turnSoundOnOff()
+playGame()
+checkLeaderboard()
<<control>>
GameMenuController
-time : Timer
+getHelp()
+showSkinStore()
+turnSoundOnOff()
+playGame()
+checkLeaderboard()
+display()
<<entity>>
Background
-time : Timer
-backgroundImage : Texture
+createBackground()
<<entity>>
Database
-leaderboard : String
+getLeaderboard() : String
+setLeaderboard(leaderboard) : void
<<entity>>
User
-username : String
-coin : Integer
-skinImage : Texture


<<boundary>>
HelpPage
+returnToMainMenu()
<<control>>
HelpPageController
+display()
+returnToMainMenu()
<<entity>>
Help
-message : String


<<boundary>>
LeaderboardPage
+returnToMainMenu()
<<control>>
LeaderboardController
+showLeaderboard()
+returnToMainMenu()
<<entity>>
Database
-leaderboard : String
+getLeaderboard() : String
+setLeaderboard(leaderboard) : void
<<entity>>
User
-username : String
-score : Integer
-highScore : Integer

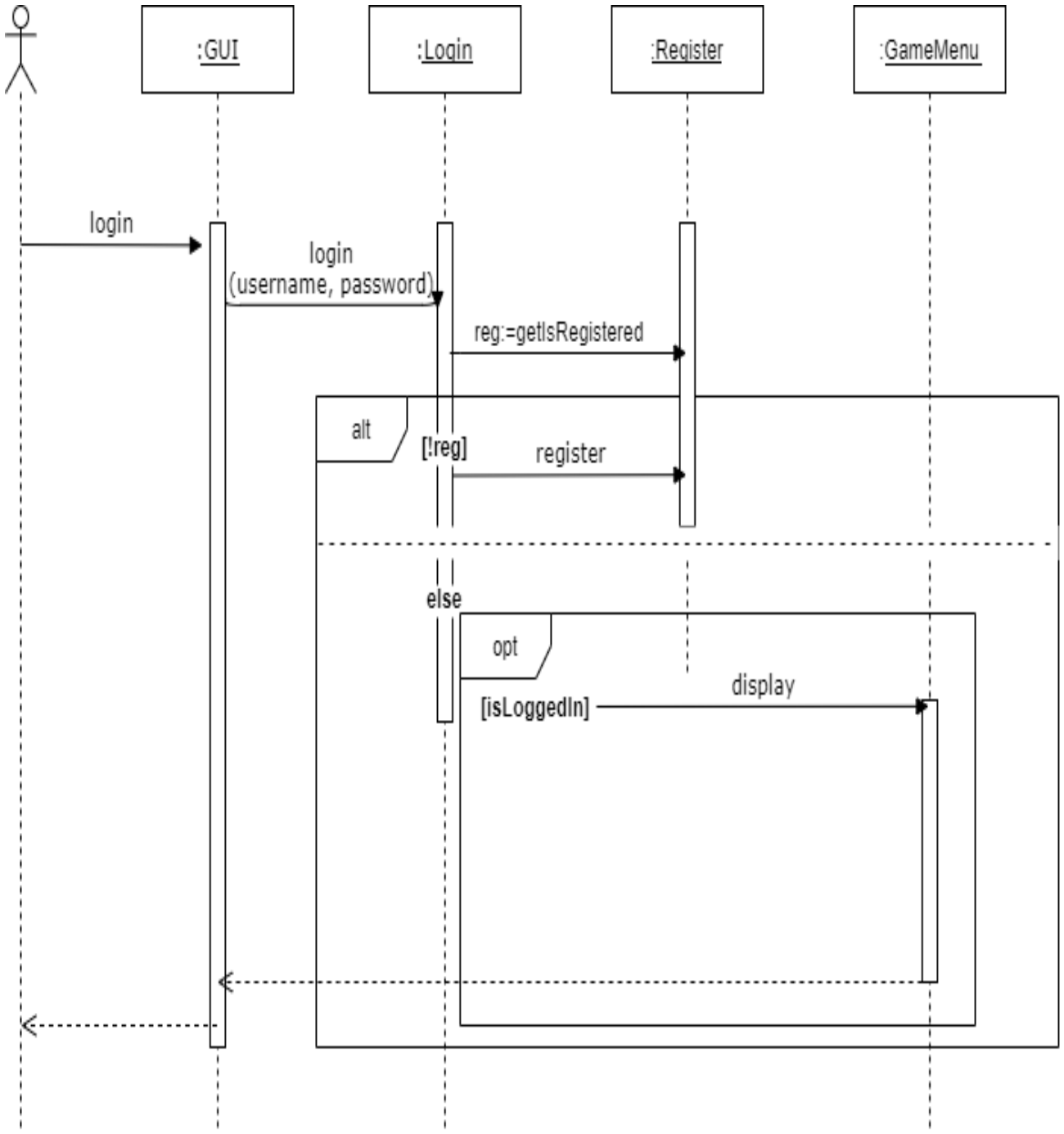

<<boundary>>
SkinStorePage
+purchaseSkins()
+useSkin()
+returnToMainMenu()
<<control>>
SkinStoreController
+purchaseSkin()
+useSkin()
+showAllSkins()
+returnToMainMenu()
<<entity>>
User
-username : String
-password : String
-coin : Integer
-skinImage : Texture
-purchasedSkins : Integer[]
+getUsername() : String
+setUsername(username) : void
+getPassword() : String
+setPassword(password) : void
+getCoin() : Integer
+setCoin(coin) : void
+getPurchasedSkins() : Integer[]
+setPurchasedSkins(purchasedSkins) : void
<<entity>>
SkinStore
-purchasedSkins : Integer[]
-skinImage : Texture
-allSkins : Integer[]
+getPurchasedSkins() : Integer[]
+setPurchasedSkins(purchasedSkins) : void

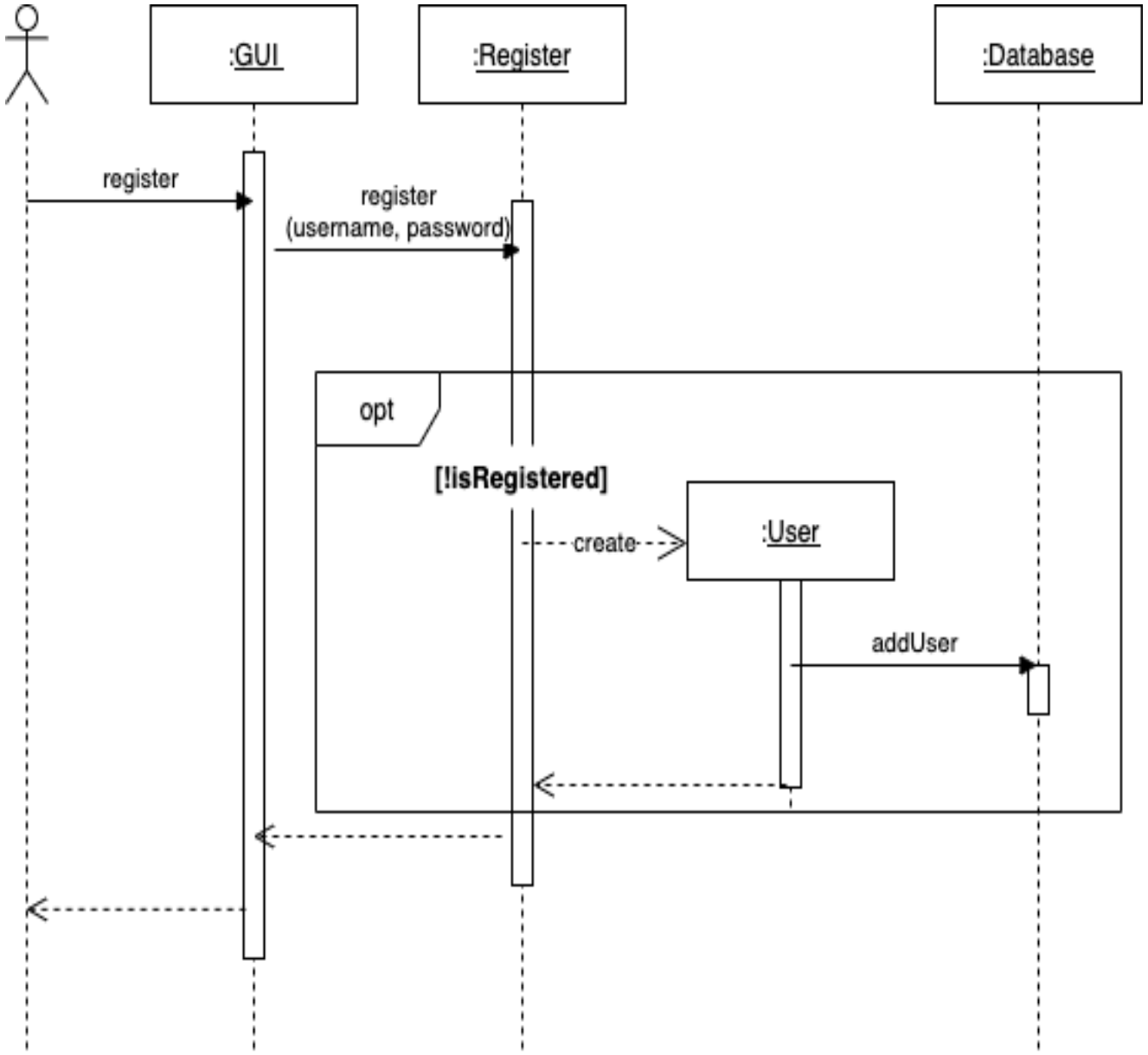

<<entity>>
Audio
-soundState : Integer
<<entity>>
CovidBlock
-coordinateX : Double
-coordinateY : Double
-time : Timer
-height : Double
-speed : Double
-blockImage : Texture
-damage : Double
<<entity>>
Item
-healthIncrease : Double
-immortality : Boolean
-time : Timer
-itemImage : Texture
<<boundary>>
PlayGame
+playGame()
+pauseGame()
<<control>>
GameSystem
-time : Timer
+setBackgroundPlayer()
+setScoreCoin()
+setVirusItems()
+setHealthPercentage()
+startGame()
+pauseGame()
+playGame()
<<entity>>
Background
-time : Timer
-backgroundImage : Texture
+createBackground()
<<entity>>
Character
-isDead : Boolean
-characterImage : Texture
-coordinateX : Double
-coordinateY : Double
-isHit : Integer
-health : Integer
+controlCharacter()
<<entity>>
User
-username : String
-coin : Integer
-skinImage : Texture

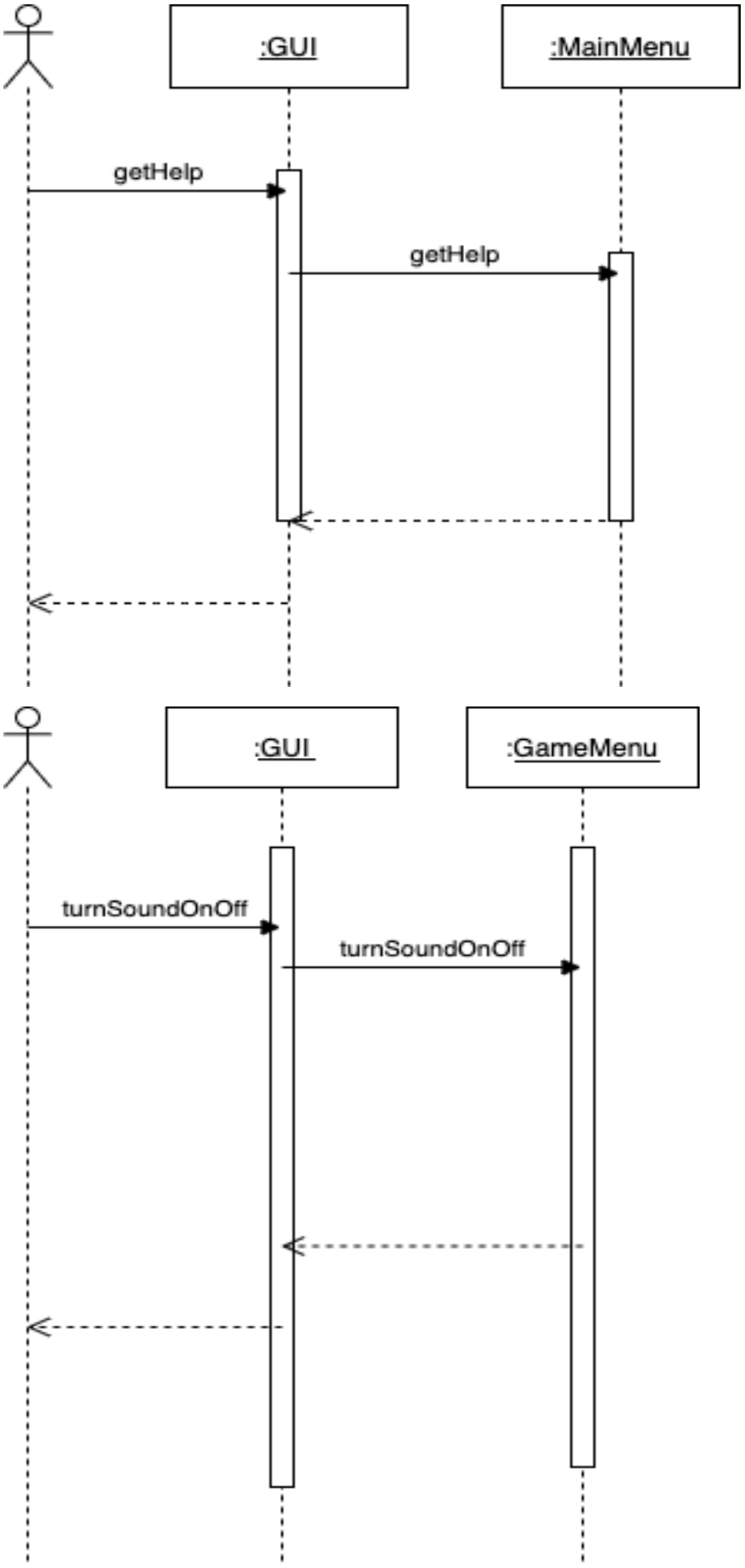

<<boundary>>
EndMenuPage
+retryGame()
+checkLeaderboard()
+returnToMainMenu()
<<control>>
EndMenuController
+retryGame()
+checkLeaderboard()
+updateLeaderboard()
+updateHighScore()
+display()
+returnToMainMenu()
<<entity>>
User
-username : String
-highScore : Integer
-coin : Integer
-score : Integer
<<entity>>
Database
-leaderboard : String
+getLeaderboard() : String
+setLeaderboard(leaderboard) : void

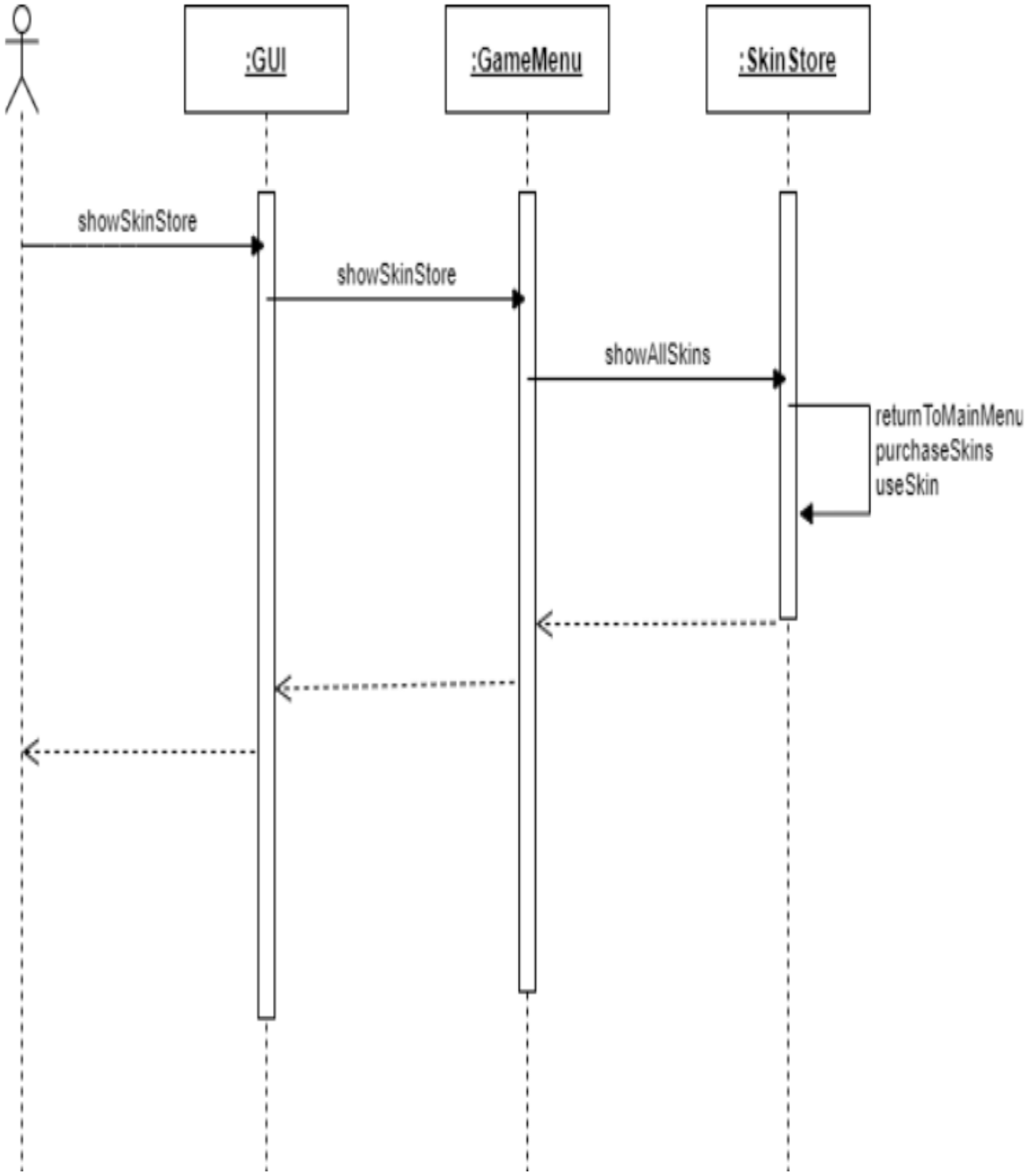
3. Dynamic Models

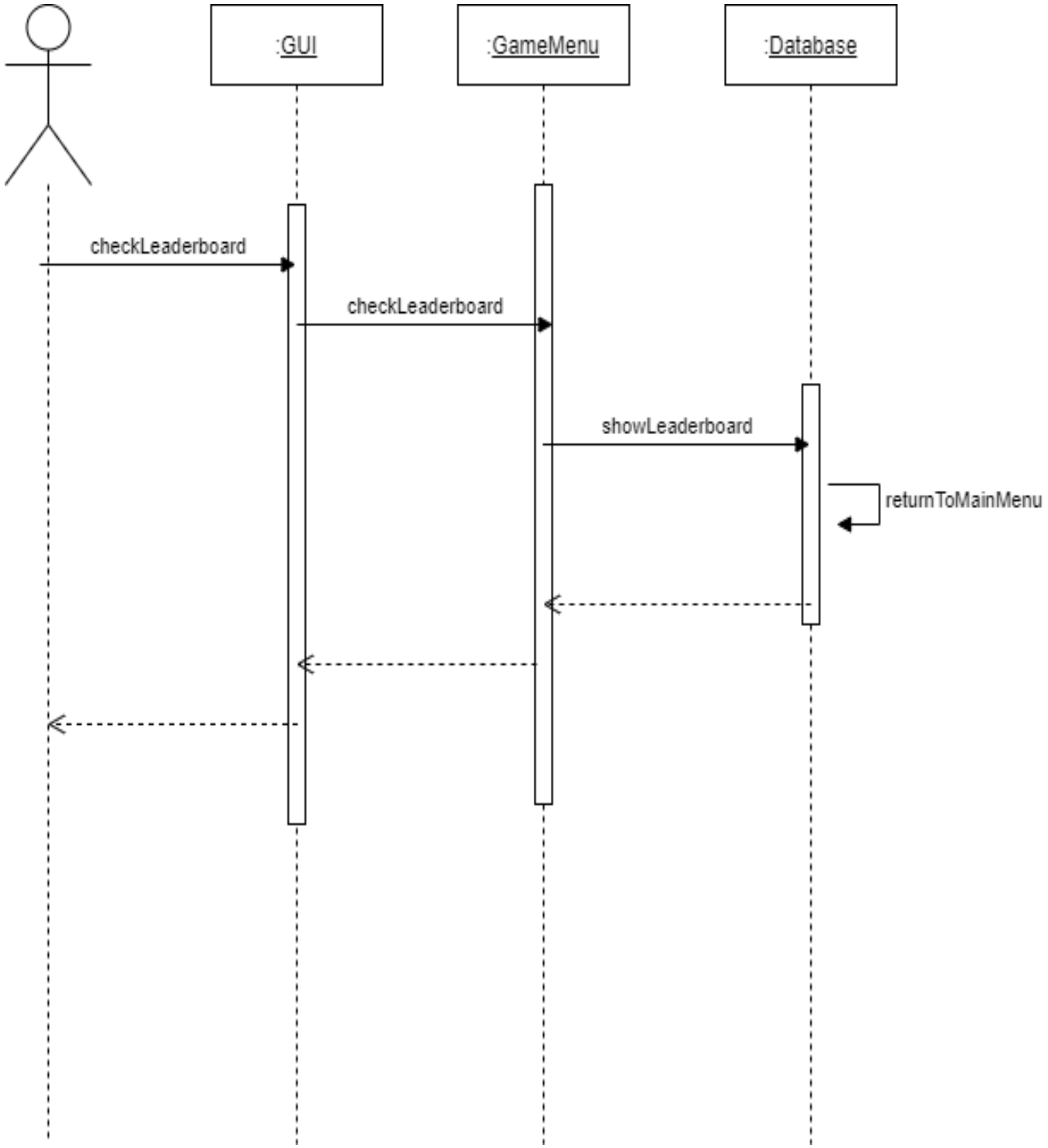
3.1. Sequence Diagrams

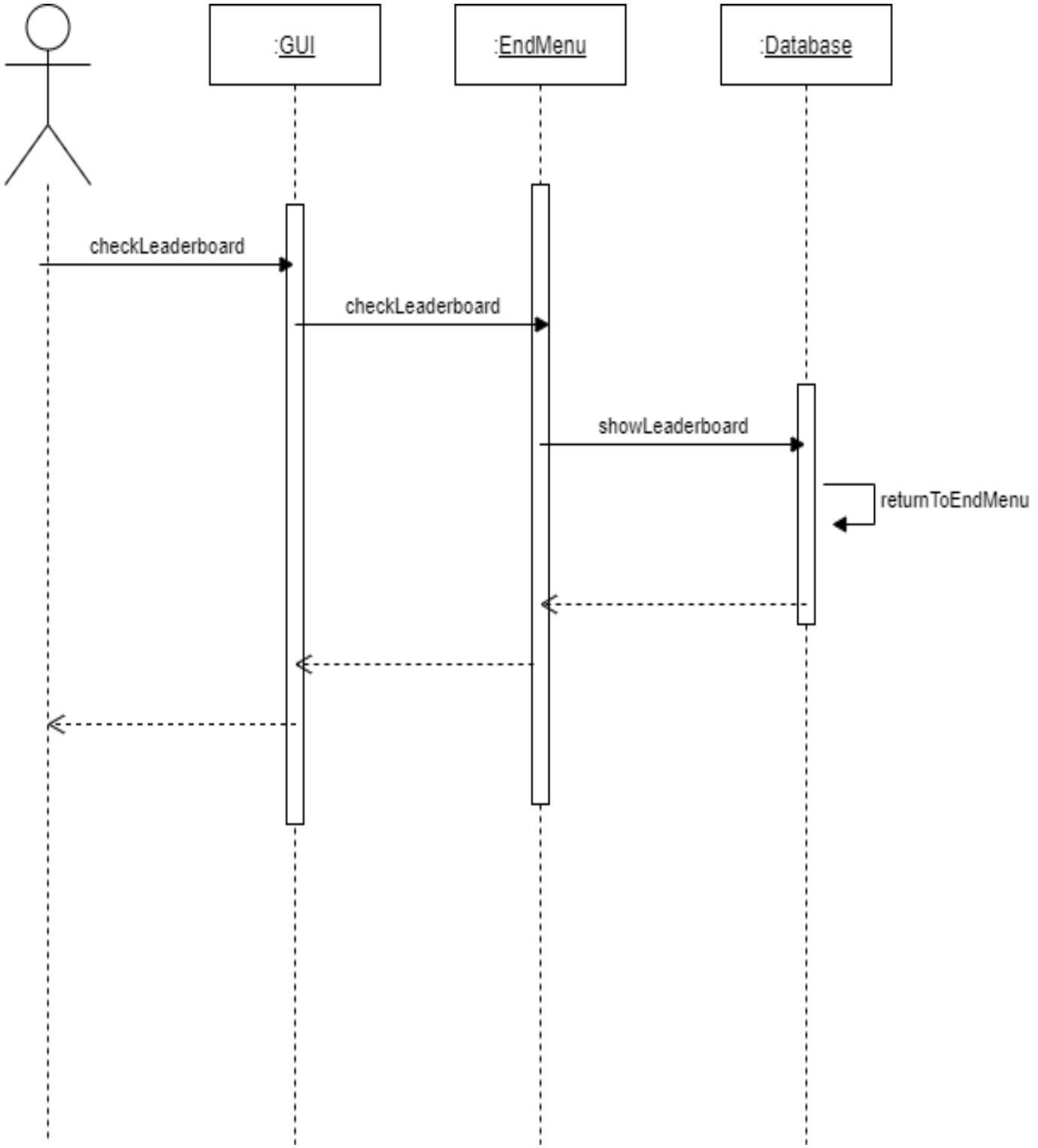


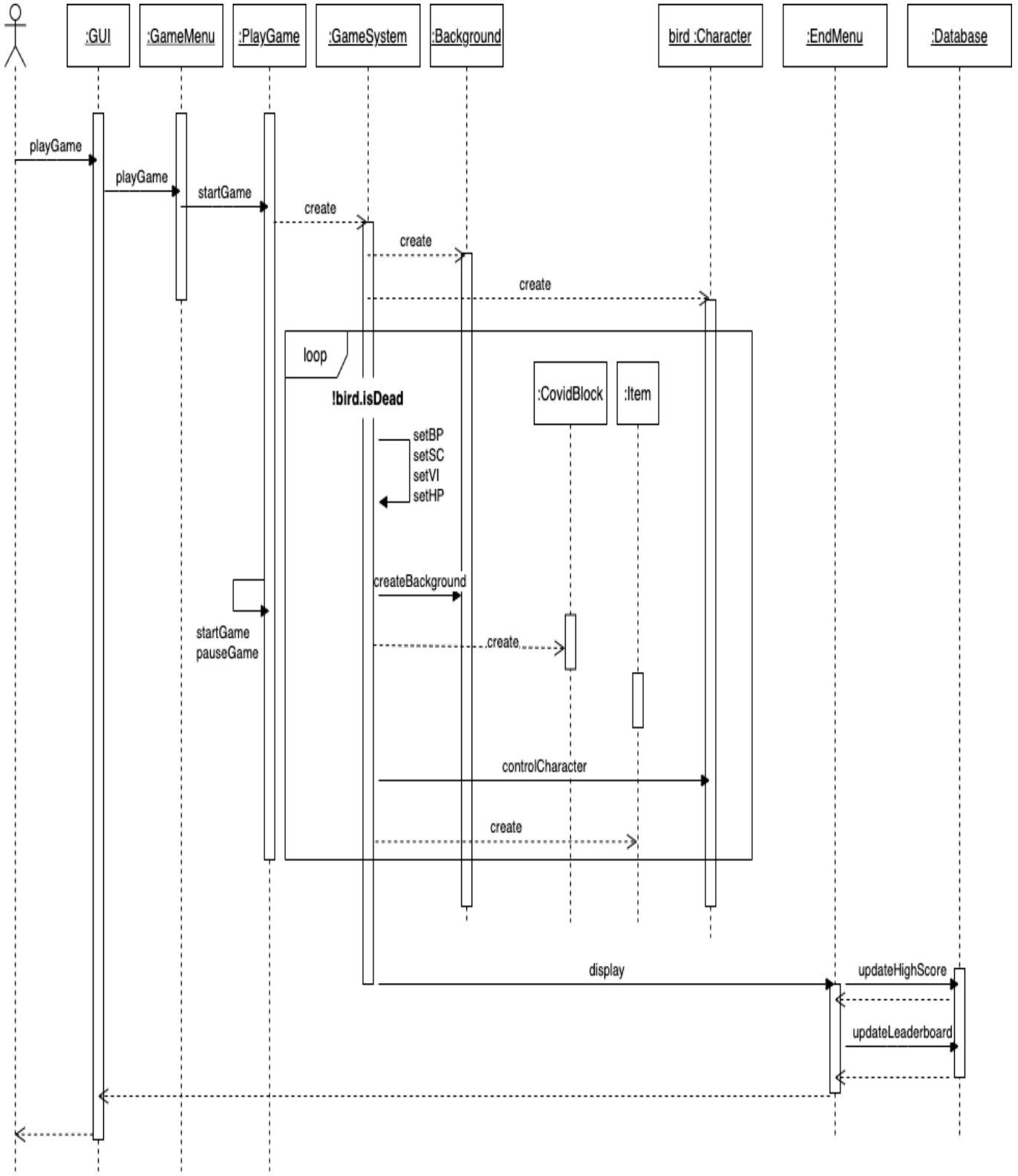

:GUI
:Register
:Database
register
register
(username, password)
opt
[!isRegistered]
:User
create
addUser

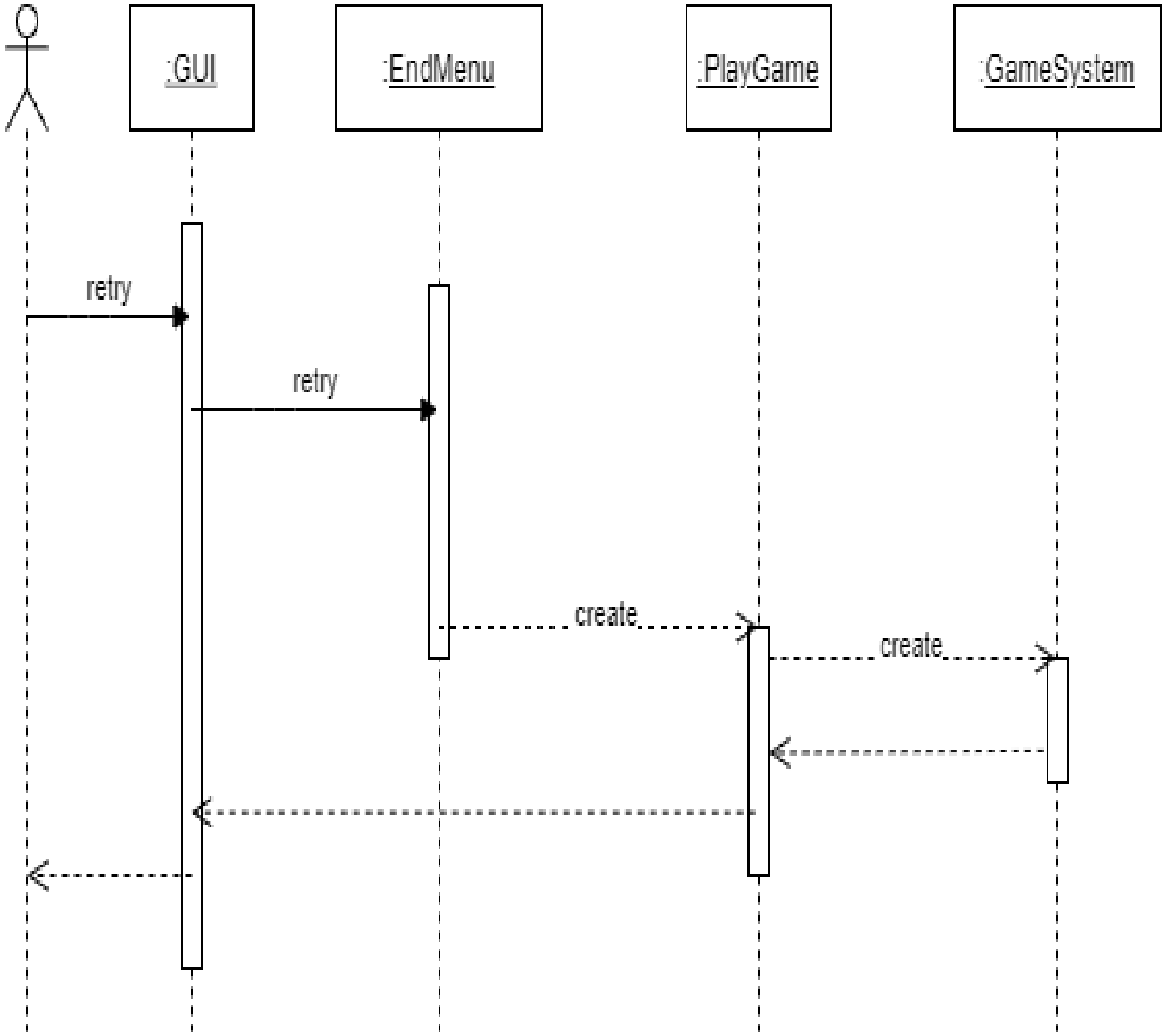

:GUI
:MainMenu
getHelp
getHelp
:GUI
:GameMenu
turnSoundOnOff
turnSoundOnOff

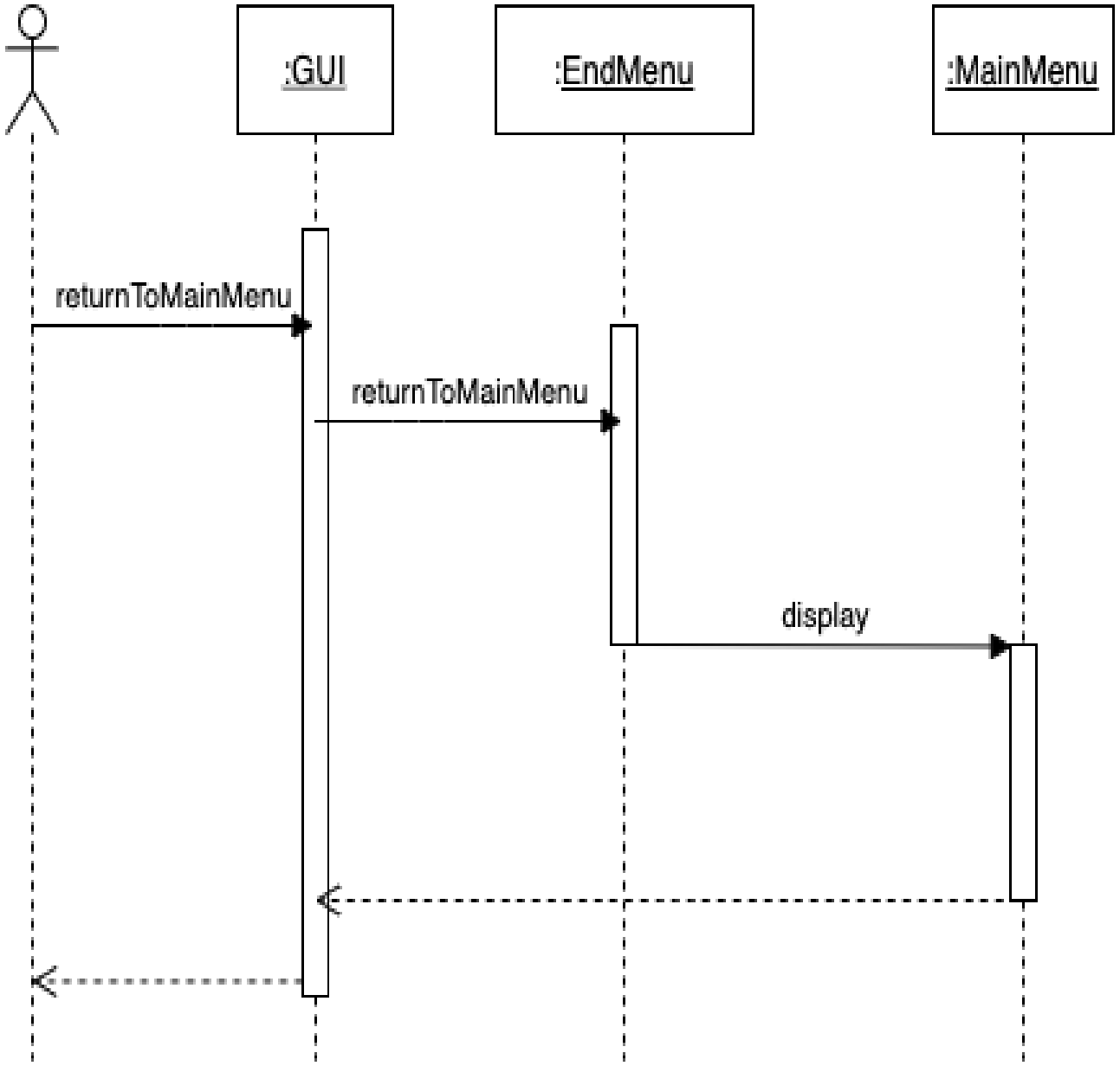

:GUI
:GameMenu
:SkinStore
showSkinStore
showSkinStore
showAllSkins
returnToMainMenu
purchaseSkins
useSkin


:GUI
:GameMenu
:Database
checkLeaderboard
checkLeaderboard
showLeaderboard
returnToMainMenu

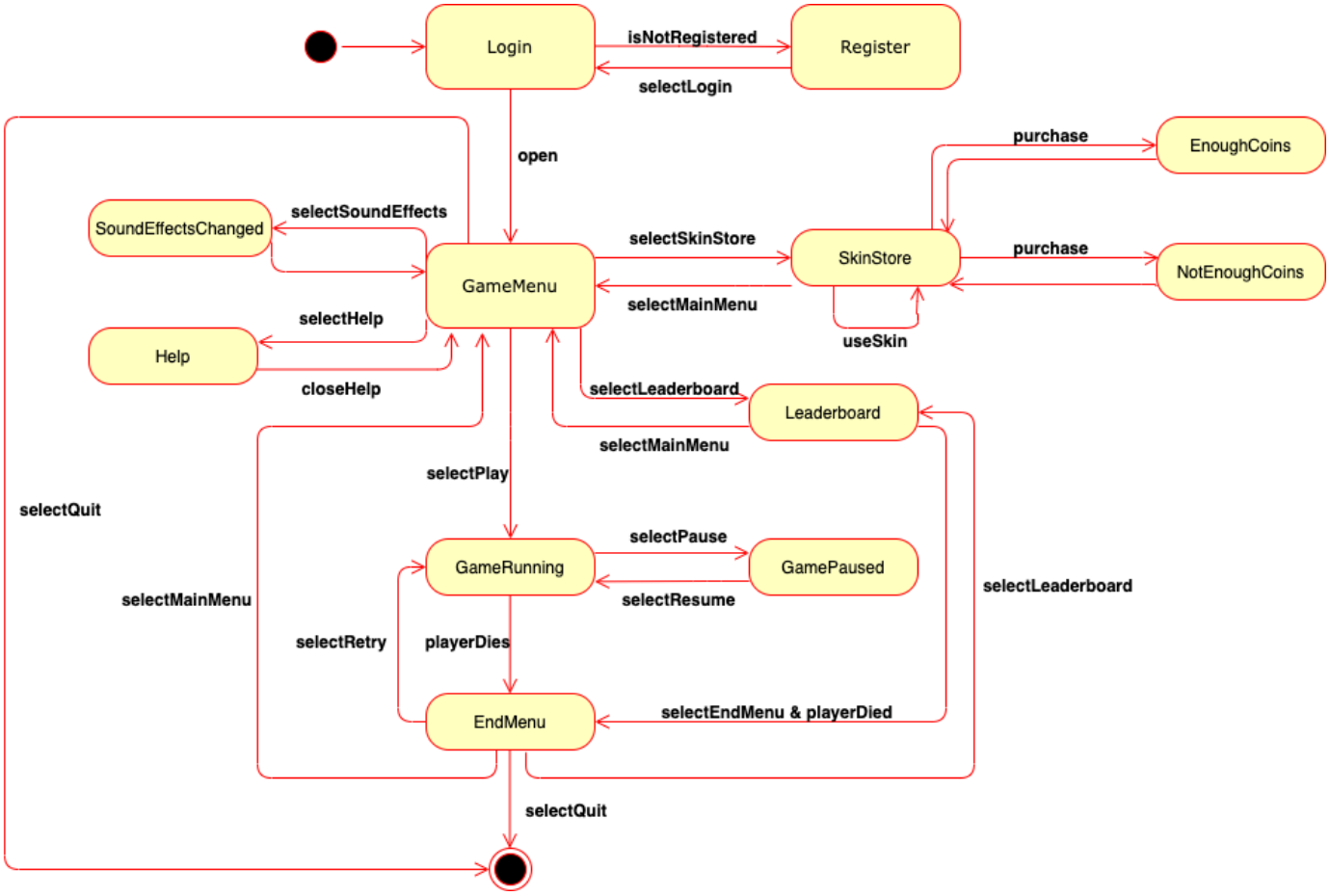

:GUI
:EndMenu
:Database
checkLeaderboard
checkLeaderboard
showLeaderboard
returnToEndMenu


:GUI
:GameMenu
:PlayGame
:GameSystem
:Background
bird :Character
:EndMenu
:Database
playGame
playGame
startGame
create
create
create
loop
!bird.isDead
:CovidBlock
:Item
setBP
setSC
setVI
setHP
createBackground
startGame
pauseGame
create
controlCharacter
create
display
updateHighScore
updateLeaderboard

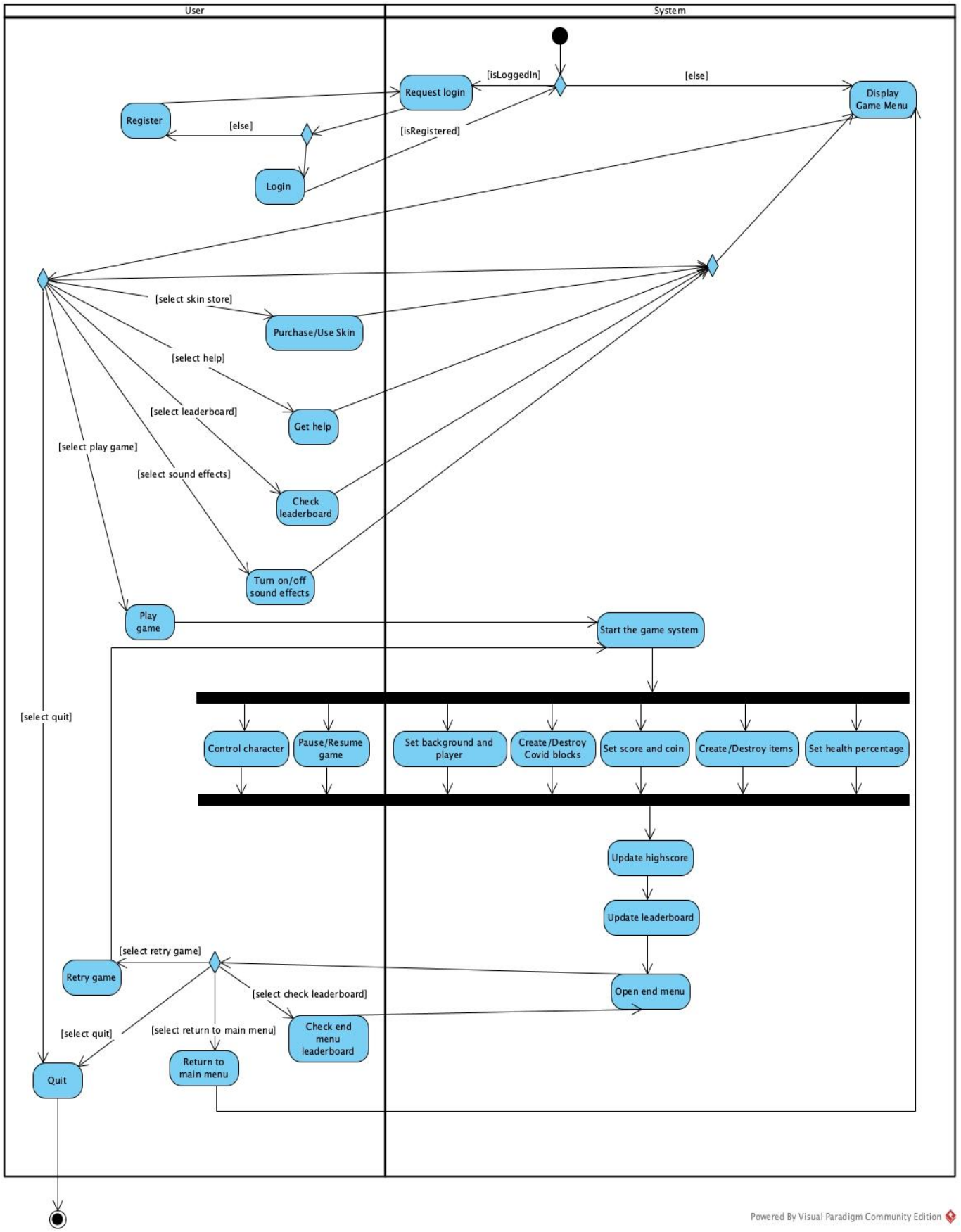

:GUI
:EndMenu
:PlayGame
:GameSystem
retry
retry
create
create


:GUI
:EndMenu
:MainMenu
returnToMainMenu
returnToMainMenu
display

3.2. State Diagram

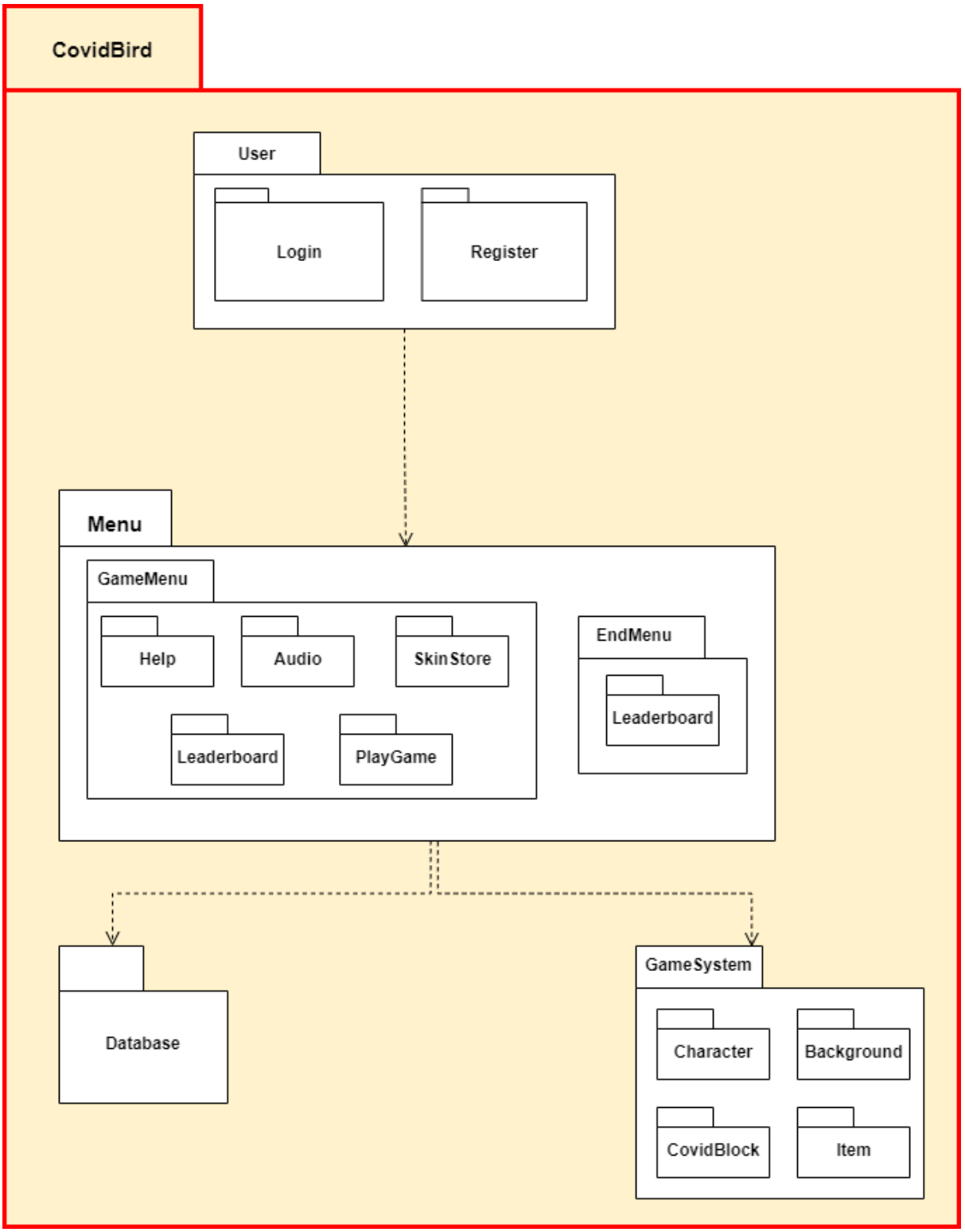


3.3. Activity Diagram

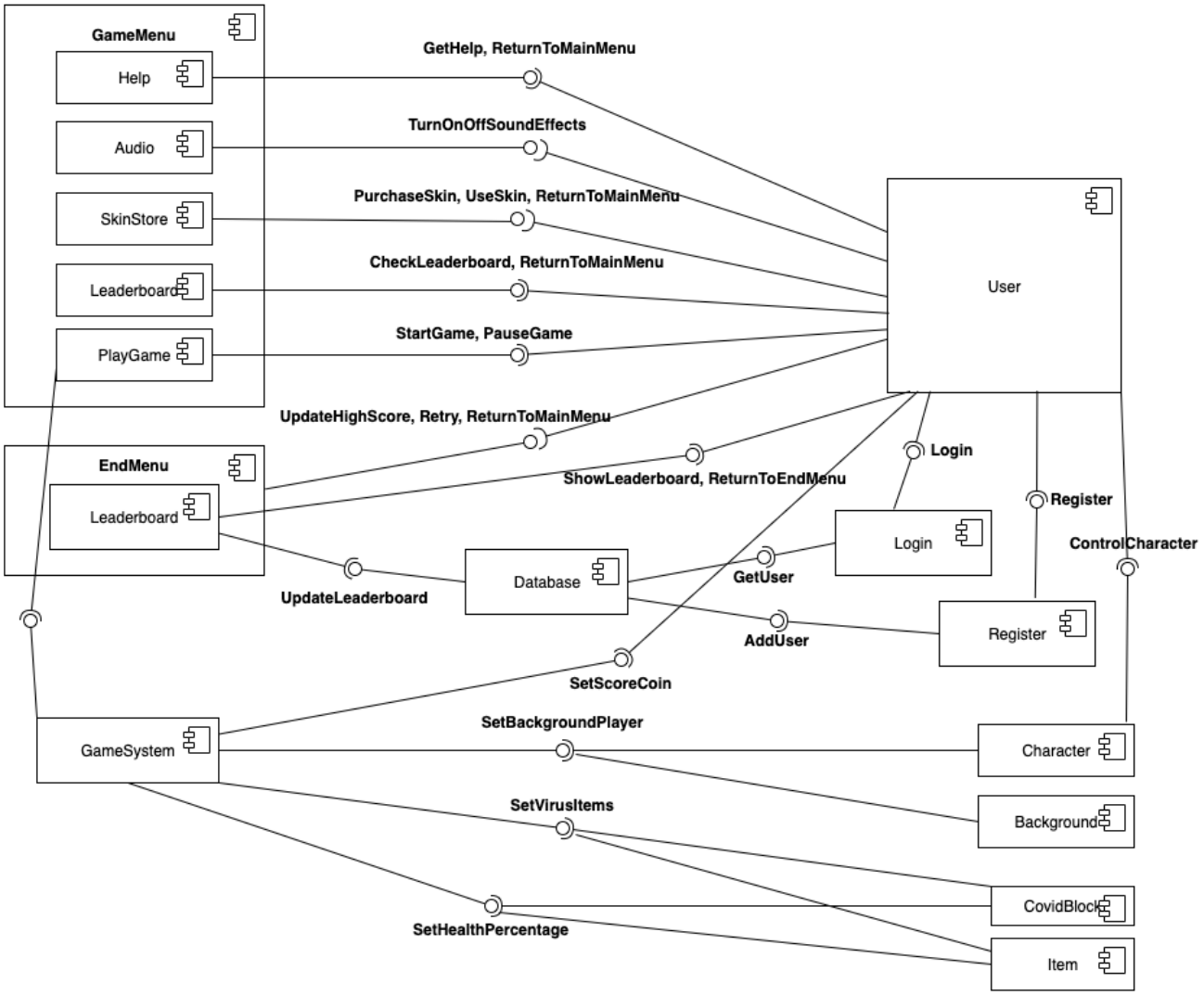


4. Software Architecture

4.1. UML Package Diagram



4.2. UML Component Diagram



GameMenu:
State:
bool Audio::soundState := soundOn;
Texture SkinStore::chosenSkin := User.characterImage;
int User::purchasedSkins[N] := 0;
Services:
provides Help::getHelp()
pre: state == "MainMenu";
post: state == "Help";

provides Help::returnToMainMenu()
pre: state == "Help";
post: state == "MainMenu";

provides Audio::turnOnOffSoundEffects()
pre: state == "MainMenu";
post: soundState == soundOn || soundState == soundOff;

provides SkinStore::purchaseSkin()
pre: state == "SkinStore";
post: User.coin -= purchasedSkinCost &&
User.purchasedSkins[@] == skinID;

provides SkinStore::useSkin()
pre: state == "SkinStore";
post: User.characterImage == chosenSkin;

provides SkinStore::returnToMainMenu()
pre: state == "SkinStore";
post: state == "MainMenu";

provides Leaderboard::checkLeaderboard()
pre: state == "MainMenu";
post: state == "Leaderboard";

provides Leaderboard::returnToMainMenu()
pre: state == "Leaderboard";
post: state == "MainMenu";

provides PlayGame::startGame()
pre: state == "MainMenu";
post: state == "PlayGame";

provides PlayGame::pauseGame()
pre: state == "PlayGame" || state == "PauseGame";
post: state == "PauseGame" || state == "PlayGame";

Database:
State:
string currentUsername := "";
string currentPassword := "";
User CurrentUser(currentUsername, currentPassword);
Services:
provides updateLeaderboard()
pre: Character.isDead == true;
post: CurrentUser.highScore == User.highScore;
provides getUser()
pre: isLoggedIn == false;
post: return(User.info);
provides addUser()
pre: isRegistered == false;
post: currentUsername == Register.username &&
currentPassword == Register.password;

EndMenu:
Services:
provides updateHighScore()
pre: Character.isDead == true && User.score >= User.highScore;
post: User.highScore == User.score;

provides retry()
pre: state == "EndMenu";
post: state == "PlayGame";

provides returnToMainMenu()
pre: state == "EndMenu"
post: state == "MainMenu"

GameSystem:
State:
int score := 0;
int coin := 0;
int currentCoordinateY := random;
Services:
provides setScoreCoin()
pre: state == "playGame";
post: User.score == GameSystem.score && User.coin == GameSystem.coin;

provides setBackgroundPlayer()
pre: state == "playGame";
post: (Background.backgroundImage == "nightbackground.png" ll "daybackground.png") &&
(Character.coordinateX == currentCoordinateX && Character.coordinateY == currentCoordinateY);

provides setVirusItems()
pre: state == "playGame";
post: CovidBlock.coordinateY == currentCoordinateY;

provides setHealthPercentage()
pre: state == "playGame";
post: Character.health -= CovidBlock.damage ll (Character.health += Item.healthIncrease ll
Character.immortality == Item.immortality);

Leaderboard:
State:
bool leaderboardPressed := false;
Services:
provides showLeaderboard()
pre: state == "EndMenu" && leaderboardPressed == true;
post: state == "Leaderboard";

provides returnToEndMenu()
pre: state == "Leaderboard";
post: state == "EndMenu";

requires updateLeaderboard()
pre: state == "EndMenu";
post: Database.CurrentUser.highScore == User.highScore;

Item:
State:
int coordinateX := random;
int coordinateY := random;
double healthIncrease := ItemType.health;
bool immortality := ItemType.immortality;
Services:
required setVirusItems()
pre: Character.isDead == false;
post: currentCoordinateX == coordinateX &&
currentCoordinateY == coordinateY;
required setHealthPercentage()
pre: Character.isDead == false;
post: Character.health += healthIncrease &&
Character.immortality == immortality;

CovidBlock:
State:
int coordinateY := random;
double damage := CovidType.damage;
Services:
required setVirusItems()
pre: Character.isDead == false;
post: currentCoordinateY == coordinateY;
required setHealthPercentage()
pre: Character.isDead == false;
post: Character.health -= damage;

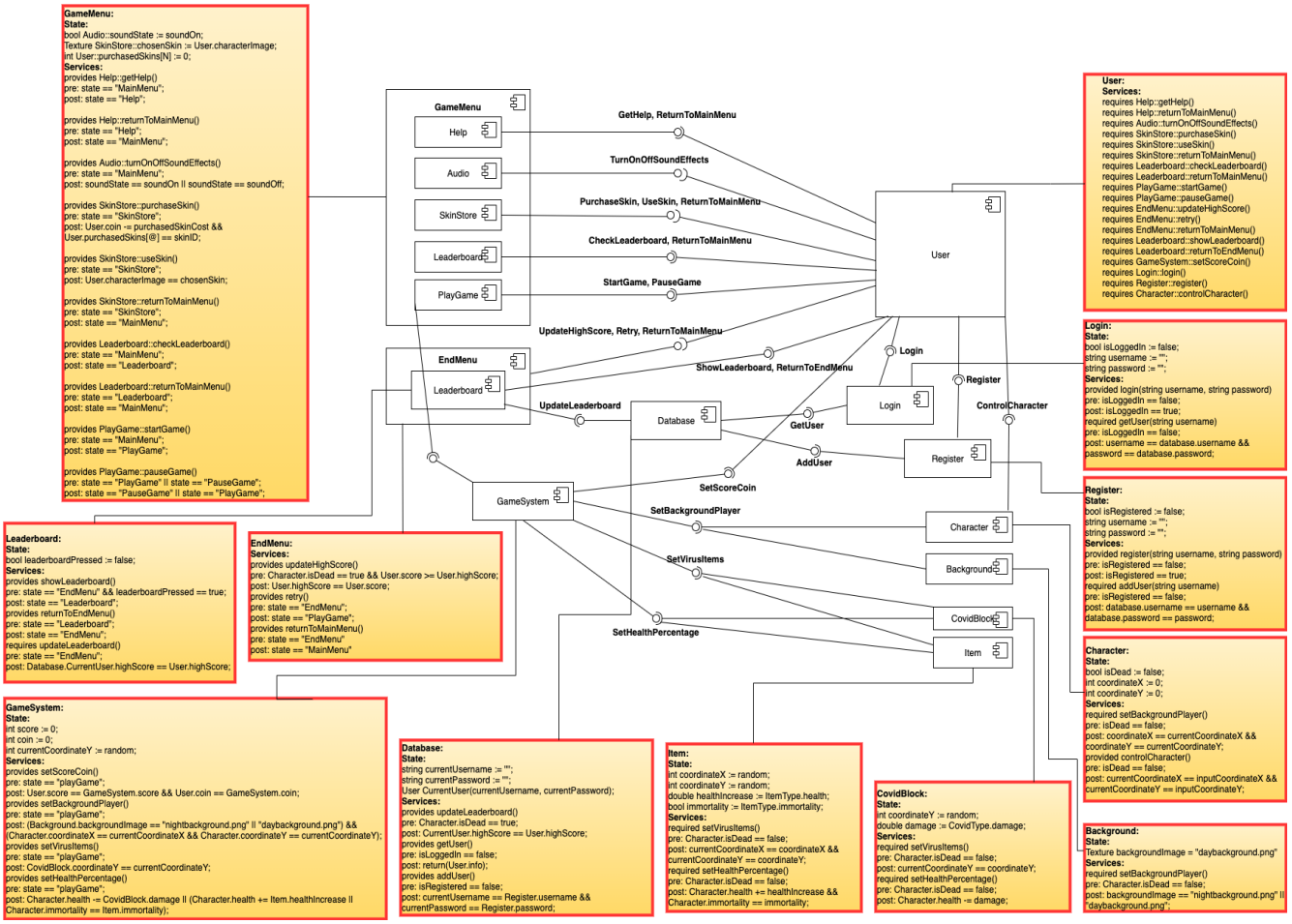
Character:
State:
bool isDead := false;
int coordinateX := 0;
int coordinateY := 0;
Services:
required setBackgroundPlayer()
pre: isDead == false;
post: coordinateX == currentCoordinateX &&
coordinateY == currentCoordinateY;
provided controlCharacter()
pre: isDead == false;
post: currentCoordinateX == inputCoordinateX &&
currentCoordinateY == inputCoordinateY;

Background:
State:
Texture backgroundImage = "daybackground.png"
Services:
required setBackgroundPlayer()
pre: Character.isDead == false;
post: backgroundImage == "nightbackground.png" ||
"daybackground.png";

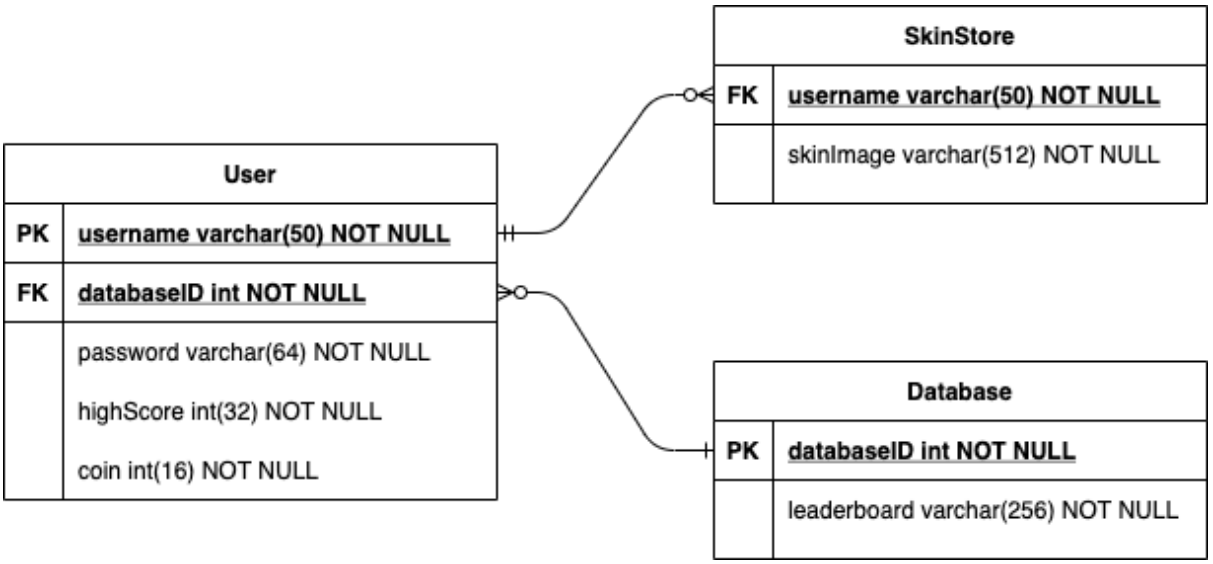
User:
Services:
requires Help::getHelp()
requires Help::returnToMainMenu()
requires Audio::turnOnOffSoundEffects()
requires SkinStore::purchaseSkin()
requires SkinStore::useSkin()
requires SkinStore::returnToMainMenu()
requires Leaderboard::checkLeaderboard()
requires Leaderboard::returnToMainMenu()
requires PlayGame::startGame()
requires PlayGame::pauseGame()
requires EndMenu::updateHighScore()
requires EndMenu::retry()
requires EndMenu::returnToMainMenu()
requires Leaderboard::showLeaderboard()
requires Leaderboard::returnToEndMenu()
requires GameSystem::setScoreCoin()
requires Login::login()
requires Register::register()
requires Character::controlCharacter()

Login:
State:
bool isLoggedIn := false;
string username := "";
string password := "";
Services:
provided login(string username, string password)
pre: isLoggedIn == false;
post: isLoggedIn == true;
required getUser(string username)
pre: isLoggedIn == false;
post: username == database.username &&
password == database.password;

Register:
State:
bool isRegistered := false;
string username := "";
string password := "";
Services:
provided register(string username, string
password)
pre: isRegistered == false;
post: isRegistered == true;
required addUser(string username)
pre: isRegistered == false;
post: database.username == username &&
database.password == password;


GameMenu:
State:
bool Audio::soundState := soundOn;
Texture SkinStore::chosenSkin := User.characterImage;
int User::purchasedSkins[N] := 0;
Services:
provides Help::getHelp()
pre: state == "MainMenu";
post: state == "Help";
provides Help::returnToMainMenu()
pre: state == "Help";
post: state == "MainMenu";
provides Audio::turnOnOffSoundEffects()
pre: state == "MainMenu";
post: soundState == soundOn || soundState == soundOff;
provides SkinStore::purchaseSkin()
pre: state == "SkinStore";
post: User.coin -= purchasedSkinCost &&
User.purchasedSkins[@] == skinID;
provides SkinStore::useSkin()
pre: state == "SkinStore";
post: User.characterImage == chosenSkin;
provides SkinStore::returnToMainMenu()
pre: state == "SkinStore";
post: state == "MainMenu";
provides Leaderboard::checkLeaderboard()
pre: state == "MainMenu";
post: state == "Leaderboard";
provides Leaderboard::returnToMainMenu()
pre: state == "Leaderboard";
post: state == "MainMenu";
provides PlayGame::startGame()
pre: state == "MainMenu";
post: state == "PlayGame";
provides PlayGame::pauseGame()
pre: state == "PlayGame" || state == "PauseGame";
post: state == "PauseGame" || state == "PlayGame";
GameMenu
Help
Audio
SkinStore
Leaderboard
PlayGame
GetHelp, ReturnToMainMenu
TurnOnOffSoundEffects
PurchaseSkin, UseSkin, ReturnToMainMenu
CheckLeaderboard, ReturnToMainMenu
StartGame, PauseGame
UpdateHighScore, Retry, ReturnToMainMenu
User
User:
Services:
requires Help::getHelp()
requires Help::returnToMainMenu()
requires Audio::turnOnOffSoundEffects()
requires SkinStore::purchaseSkin()
requires SkinStore::useSkin()
requires SkinStore::returnToMainMenu()
requires Leaderboard::checkLeaderboard()
requires Leaderboard::returnToMainMenu()
requires PlayGame::startGame()
requires PlayGame::pauseGame()
requires EndMenu::updateHighScore()
requires EndMenu::retry()
requires EndMenu::returnToMainMenu()
requires Leaderboard::showLeaderboard()
requires Leaderboard::returnToEndMenu()
requires GameSystem::setScoreCoin()
requires Login::login()
requires Register::register()
requires Character::controlCharacter()
EndMenu
Leaderboard
ShowLeaderboard, ReturnToEndMenu
Login
Register
ControlCharacter
UpdateLeaderboard
Database
GetUser
AddUser
Login:
State:
bool isLoggedIn := false;
string username := "";
string password := "";
Services:
provided login(string username, string password)
pre: isLoggedIn == false;
post: isLoggedIn == true;
required getUser(string username)
pre: isLoggedIn == false;
post: username == database.username &&
password == database.password;
Register
SetScoreCoin
GameSystem
SetBackgroundPlayer
SetVirusItems
SetHealthPercentage
Character
Background
CovidBlock
Item
Register:
State:
bool isRegistered := false;
string username := "";
string password := "";
Services:
provided register(string username, string password)
pre: isRegistered == false;
post: isRegistered == true;
required addUser(string username)
pre: isRegistered == false;
post: database.username == username &&
database.password == password;
Leaderboard:
State:
bool leaderboardPressed := false;
Services:
provides showLeaderboard()
pre: state == "EndMenu" && leaderboardPressed == true;
post: state == "Leaderboard";
provides returnToEndMenu()
pre: state == "Leaderboard";
post: state == "EndMenu";
requires updateLeaderboard()
pre: state == "EndMenu";
post: Database.CurrentUser.highScore == User.highScore;
EndMenu:
Services:
provides updateHighScore()
pre: Character.isDead == true && User.score >= User.highScore;
post: User.highScore == User.score;
provides retry()
pre: state == "EndMenu";
post: state == "PlayGame";
provides returnToMainMenu()
pre: state == "EndMenu"
post: state == "MainMenu"
Character:
State:
bool isDead := false;
int coordinateX := 0;
int coordinateY := 0;
Services:
required setBackgroundPlayer()
pre: isDead == false;
post: coordinateX == currentCoordinateX &&
coordinateY == currentCoordinateY;
provided controlCharacter()
pre: isDead == false;
post: currentCoordinateX == inputCoordinateX &&
currentCoordinateY == inputCoordinateY;
GameSystem:
State:
int score := 0;
int coin := 0;
int currentCoordinateY := random;
Services:
provides setScoreCoin()
pre: state == "playGame";
post: User.score == GameSystem.score && User.coin == GameSystem.coin;
provides setBackgroundPlayer()
pre: state == "playGame";
post: (Background.backgroundImage == "nightbackground.png" || "daybackground.png") &&
(Character.coordinateX == currentCoordinateX && Character.coordinateY == currentCoordinateY);
provides setVirusItems()
pre: state == "playGame";
post: CovidBlock.coordinateY == currentCoordinateY;
provides setHealthPercentage()
pre: state == "playGame";
post: Character.health -= CovidBlock.damage || (Character.health += Item.healthIncrease ||
Character.immortality == Item.immortality);
Database:
State:
string currentUsername := "";
string currentPassword := "";
User CurrentUser(currentUsername, currentPassword);
Services:
provides updateLeaderboard()
pre: Character.isDead == true;
post: CurrentUser.highScore == User.highScore;
provides getUser()
pre: isLoggedIn == false;
post: return(User.info);
provides addUser()
pre: isRegistered == false;
post: currentUsername == Register.username &&
currentPassword == Register.password;
Item:
State:
int coordinateX := random;
int coordinateY := random;
double healthIncrease := ItemType.health;
bool immortality := ItemType.immortality;
Services:
required setVirusItems()
pre: Character.isDead == false;
post: currentCoordinateX == coordinateX &&
currentCoordinateY == coordinateY;
required setHealthPercentage()
pre: Character.isDead == false;
post: Character.health += healthIncrease &&
Character.immortality == immortality;
CovidBlock:
State:
int coordinateY := random;
double damage := CovidType.damage;
Services:
required setVirusItems()
pre: Character.isDead == false;
post: currentCoordinateY == coordinateY;
required setHealthPercentage()
pre: Character.isDead == false;
post: Character.health -= damage;
Background:
State:
Texture backgroundImage = "daybackground.png"
Services:
required setBackgroundPlayer()
pre: Character.isDead == false;
post: backgroundImage == "nightbackground.png" ||
"daybackground.png";

5. Entity Relationship Diagram



6. Glossary & References

6.1. Glossary

Database: A database is an organized collection of data, generally stored and accessed electronically from a computer system.

Sequence Diagram: A sequence diagram shows object interactions arranged in time sequence.

State Diagram: State diagram is a type of diagram used in computer science and related fields to describe the behavior of systems.

Activity Diagram: Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

Entity Relationship Diagram: An entity–relationship model (or ER model) describes interrelated things of interest in a specific domain of knowledge.

UML Component Diagram: In Unified Modeling Language (UML), a component diagram depicts how components are wired together to form larger components or software systems.

UML Package Diagram: The package diagram, a kind of structural diagram, shows the arrangement and organization of model elements in middle to large scale project.

SetBP: Set Background Player

SetSC: Set Score & Coin

SetVI: Set Virus Item

SetHP: Set Health Percentage

GUI: Graphical User Interface

OPT: Optional

ALT: Alternative

FK: Foreign Key

PK: Primary Key

6.2. References

[1]https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/

[2]https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

[3]https://www.visual-paradigm.com/support/documents/vpuserguide/94/2584/7191_drawingobjec.html

[4]https://app.diagrams.net/

[5]https://www.site.uottawa.ca/~tcl/seg2105/coursenotes/ClassDiagramChecklist.html

[6]https://en.wikipedia.org

[7]https://online.visual-paradigm.com/diagrams/tutorials/component-diagram-tutorial/

[8]https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning

[9]https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/

[10]https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/

[11]https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/

[12]Timothy C. Lethbridge and Robert Laganière. (2004). *Object-Oriented Software Engineering: Practical Software Development using UML and Java.* (2nd edition). McGraw Hill Higher Education.