Software Requirements Specification (SRS)
Project Daily Gambit

Team: TheAnh Tran, Zhu Lu, Matthew Svenson, ThienTran Le, Tim Truong, Ashraf Mohid (Team 9)
Authors: Zhu Lu, Matthew Svenson, ThienTran Le, Tim Truong, TheAnh Tran, Ashraf Mohid
Customer: Middle School Educators and Minors
Instructor: Dr. Daly

1 Introduction

The subsections of this SRS will include information that is pertinent to the project at hand.

In this first section, aptly named “Introduction”, there will be an overview of what the Software Requirements Specifications contain. The subsections in this introduction are as follows:

- **1.1 Purpose**
- **1.2 Scope**
- **1.3 Definitions, acronyms, and abbreviations**
- **1.4 Organization**

Section 1.1 will discuss the project’s purpose – what the software and goals are for the project, the target audience it is aimed for, and how the project can help in emphasizing those goals.

Then, Section 1.2 will go in depth of the scope of the project, detailing more on what the software is. It will discuss what tools are being used and the general limitations can be for using those tools.

Next, Section 1.3 will help in defining any and all possible shorthand mentioned in this document. For any terms that have a different meaning other than the one that is conventionally used, it will also be mentioned here.

Lastly, Section 1.4 will go through the organization of the rest of the Software Requirements Specifications, as well as a general description for what the sections will go over in their respective order.

1.1 Purpose

The purpose of this software requirements specifications document is to outline the project as well as any resources and diagrams that were used in the process of the software creation and conception for Daily Gambit. Teachers are the customers who want the software to be created as well as the development team involved in the creation of this software.

1.2 Scope

The software project that will be made will be called “Daily Gambit”. The application domain for Daily Gambit will be in HTML, CSS, and Javascript. Daily Gambit will be an edutainment game that has elements of financial responsibility and problem-solving skills to teach to younger audiences. It will provide mathematical problems to students ranging from grades 1 to 8 for them to practice and develop decision-making skills when considering how to deal with managing and selling stocks or wagering current money to make a potential profit.

1.3 Definitions, acronyms, and abbreviations

DG – Daily Gambit

SRS – Software Requirements Specifications

HTML – HyperText Markup Language; a markup language for content displayed on a web browser

CSS – Cascading Style Sheets; language for describing the presentation of other markup languages

JS – JavaScript; a markup language used to enhance website features on web-browsers

1.4 Organization

The rest of this document will contain these subsections in order: Overall Description, Product Perspective, Product Functions, User Characteristics, Constraints, Assumptions and Dependencies, Apportioning of Requirements, Specific Requirements, Modeling Requirements, Prototype, How to Run Prototype, Sample Scenarios, and References. A description of the content of every subsection is located within the Introduction subsection above if needed.

The rest of the SRS will go through the sections as follows:

- **2 Overall Description**
    - **2.1 Product Perspective**
    - **2.2 Product Functions**
    - **2.3 User Characteristics**
    - **2.4 Constraints**
    - **2.5 Assumptions and Dependencies**
    - **2.6 Apportioning of Requirements**
- **3 Specific Requirements**
- **4 Modeling Requirements**
    - **4.1 Use Cases**
    - **4.2 Class Diagram**
    - **4.3 Sequence Diagram**
    - **4.4 State Diagram**
- **5 Prototype**
    - **5.1 How to Run Prototype**
    - **5.2 Sample Scenarios**
- **6 References**
- **7 Point of Contact**

Section 2 will go more in-depth over the general product and how it will function. Section 2.1 will go over the product's perspective, showing what the context for the product was and all the interfaces that may be required for the software product. Section 2.2 goes into the functions, defining all the major functions of the software as well as any

diagrams for the product. Section 2.3 will go into the user characteristics, where the expectations about the general characteristics of the targeted users are defined. Section 2.4 will go into the product constraints, defining all the interface and software constraints. Section 2.5 is about the assumptions and dependencies subsection, where any of the assumptions for first version expectations and dependencies are for the product to run off of. Section 2.6 is the apportioning of requirements, where a discussion about requirements that are beyond the scope of the original project is placed. This subsection in particular will contain information for when future versions of the software will be released.

Then, Section 3 will go over the specific requirements of the product. This all will be based on the general analysis of the goals needed to be achieved from the product.

Next, Section 4 will document the modeling requirements and this will be where the requirements from Section 3 will be used to create several diagrams that will assist in the product development process. This will be where the use cases and some representative scenarios will be shown in Section 4.1 and will go into detail about the class, sequence, and state diagrams in Sections 4.2 – 4.4, respectively.

Following that, Section 5 will go over the product prototype, and will describe how to run and execute the product in Section 5.1, as well as a sample scenario for how the target demographic is to play the product in Section 5.2.

After, Section 6 will go into the references used to create the data used in the game as well as any other sources that have gone into inspiring the goals for the product.

Lastly, Section 7 will show how to contact the development team, in the case there are any additional questions that aren't covered in the scope of this SRS.

2 Overall Description

2.1 Product Perspective

The context for DG is to create an “edutainment” game that can be played by any child between grades 1 to 8. The game is a standalone that only requires the website domain to run, it is part of the bigger system of education and the relationship between teachers and students. In terms of interface constraints, the user will need a monitor, keyboard, and mouse to be able to play the game.

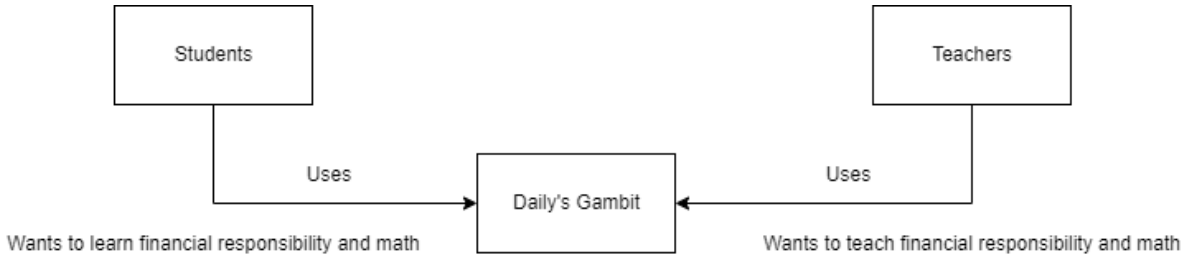


Figure 2.1.1: a general example of the student-teacher relationship for DG

2.2 Product Functions

One of the major functions that the software will perform will be educating teens with questions while providing a fun time. There are 3 key components in our game:

- the investment system
- the question function
- “gambling”/risk component

The investment system will allow the user to pick which stock they want to invest their money in.

The start question/next turn function will require the user to answer a question and if they get it correct they will receive a reward for it.

Lastly, the gamble/risk function but where the user can choose to risk what they have for greater rewards.

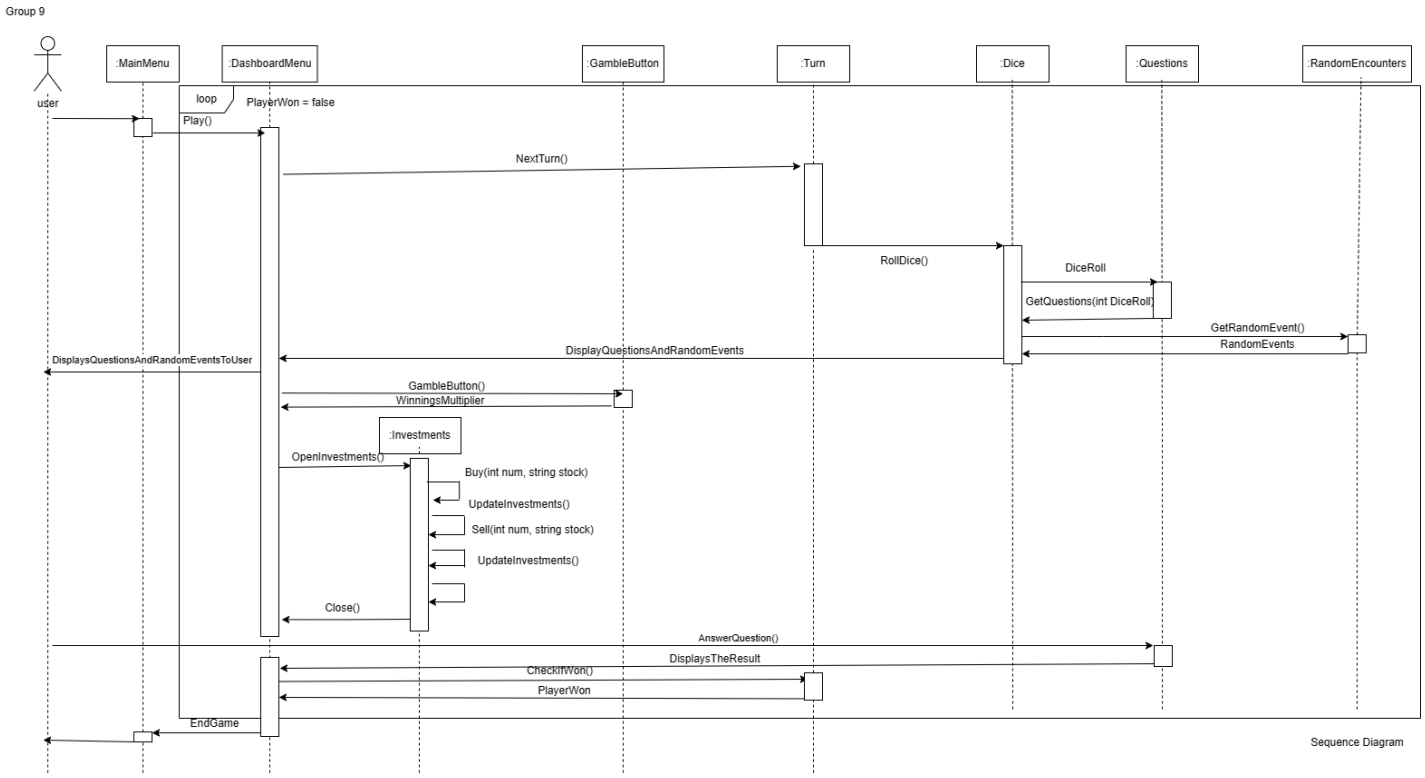


Figure 2.2.1: the sequence diagram for the investment system

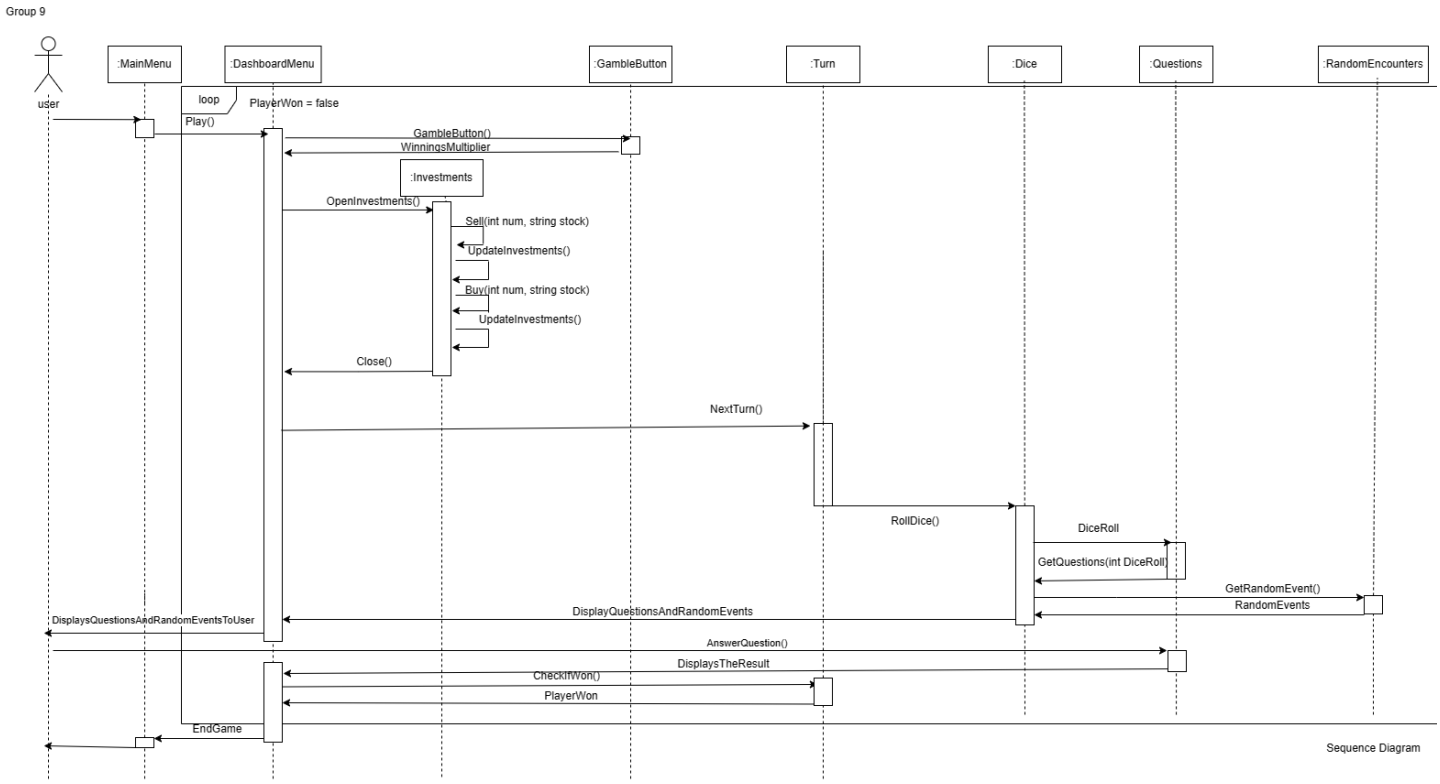


Figure 2.2.2: the sequence diagram for the question system

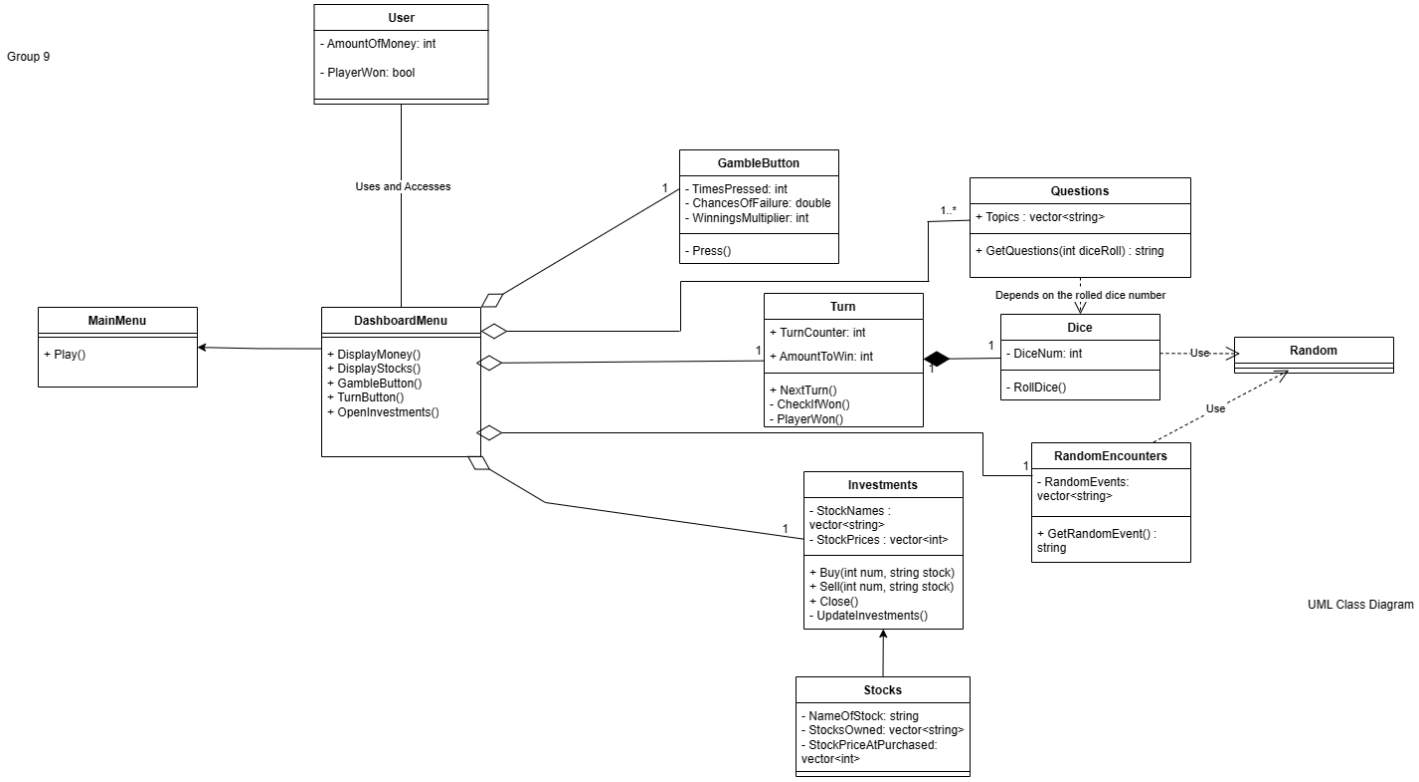


Figure 2.2.3: the class diagram for the overall game play of the product

For the high-level class diagram the main goal is to improve the math and decision-making skill of the user who will be playing the game. This is done through the goal and task of the game which allows the user to get educated with risk and probability. We want to provide a visually engaging game with the ability to let the user choose what to invest in and test their math skill and understanding of risk.

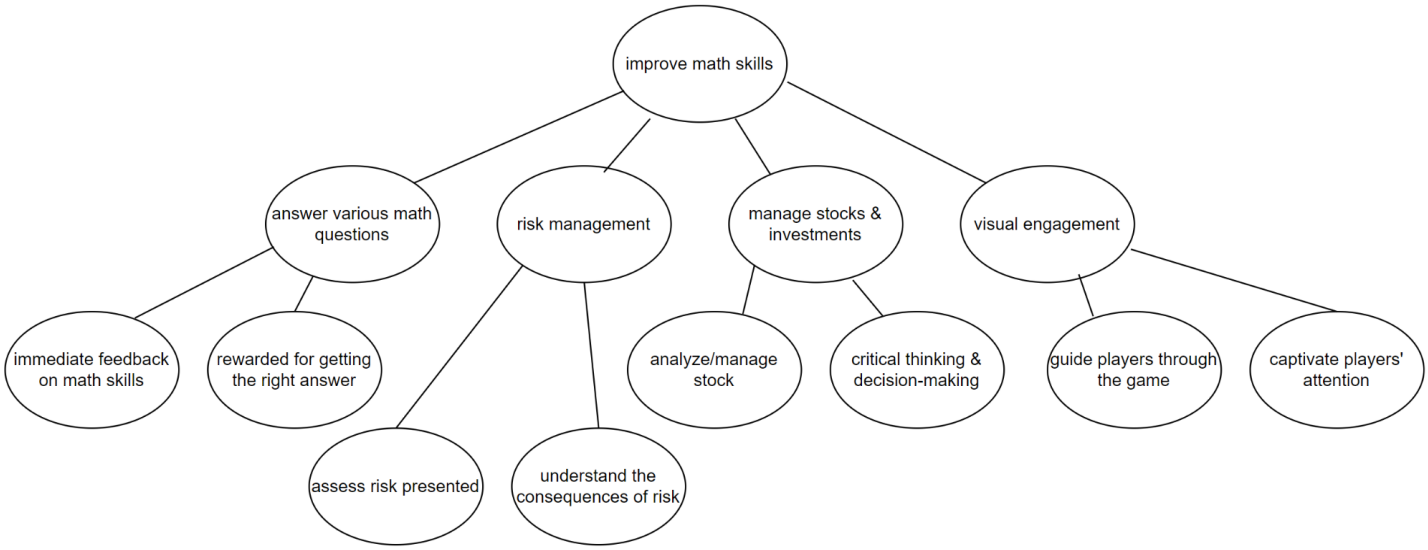


Figure 2.2.4: the high-level class diagram to describe the main goal for the product

2.3 User Characteristics

We are expecting the user to be a middle schooler male or female who knows a little pre-algebra. They might be very interested in games that will help pass the time at school. We are just aiming for the typical public middle school student who has a passion for learning more about math.

2.4 Constraints

2.4.1 Development Constraints

The development of this game shall adhere to the absolute minimum budget partitioned by the manager. The project shall be completed within the specified timeline. Delays may impact the overall project schedule and phases. The game shall be developed using JavaScript. Deviation from the predefined technology stack must be justified and approved by the project manager.

2.4.2 Legal and Ethical Constraints

The game will adhere to all relevant laws and regulations in the jurisdictions where it is made available. The game will also not promote or encourage real-world gambling activities. It must include a mechanism to prevent excessive virtual gambling within the game to maintain a responsible and ethical gaming environment. The game may be subject to age restrictions, and appropriate measures must be implemented to ensure that it is not accessible to individuals below the legal age for such games in their respective regions. The development team must ensure that all elements of the game, including graphics, sounds, and code, do not infringe upon existing intellectual property rights. Proper licensing and permissions must be obtained for any third-party assets used.

2.4.3 Platform Compatibility

The game shall be designed to run on all operating systems supporting a major browser, *Chrome*, *Internet Explorer*, *Mozilla*, etc. The game will require an internet connection to play and will not be able to run locally. If the user-side connection is not stable, the game will not function as intended.

2.4.4 Platform Safety

The game requires a stable online connection and will make sure no user information can or will be stored or distributed. The hosting service will be secure and certified.

2.5 Assumptions and Dependencies

2.5.1 Assumptions

Users are assumed to have a basic understanding of dice games and virtual currencies. The game's design and tutorials will be developed with the expectation that the players have some familiarity with these concepts. Users will be assumed to speak English, it being the only language supported. It is assumed that the users will be playing

on devices connected to the internet and through a major web browser supporting JavaScript. A stable connection will give best results, and is assumed.

2.5.2 Dependencies

The game may depend on third-party libraries or APIs for certain functionalities, such as random number generation, financial calculations, or stock market simulations. Successful development is dependent on the expertise of the development team in the chosen technology and language (JavaScript). The further inclusion of third-party graphics, sounds, or other assets in the game depends on obtaining proper licenses and permissions. Failure to secure appropriate licenses may result in legal issues and could impact the game's development and release.

2.6 Apportioning of Requirements

Requirements will be determined by the scope of the current project. Any requirements suggested outside the scope may be subject to market analysis after the release of the game. Upon further consideration, future additions may be released based on open conversations with users and stakeholders. The development team will be made aware of the scope and requirements based on the characteristics described.

3 Specific Requirements

1. The product operates on a turn-based system.
    1.1. A turn is defined as the cycle of possible choices consisting of buying investments and pressing the button.
    1.2. A turn starts with the roll of a 6-sided die and a question based on that result.
    1.3. A turn ends when the player has chosen what to do with the money they received from answering the question.
2. The product will open with a main menu.
    2.1. The main menu will be where the player chooses to start/exit the game.
    2.2. If they choose to start the game, they will be brought to the dashboard interface.
3. The product has a main dashboard interface.
    3.1. The main dashboard will display a button to initiate the roll of the die.
    3.2. After the die has been rolled, the investment button, gambling button, and the next turn button will be made available.
4. The player must answer the end-of-turn question after they roll the die.
    4.1. Depending on the dice number, initiate a math problem that coincides with that number. Each number corresponds to a different topic in math.
    4.2. The questions will be based on the 2017 Curriculum Framework for Mathematics located in The Massachusetts Department of Education and Common Core standards website.
    4.3. Players can only proceed with the rest of their turn after the question has been answered.
5. The player can “risk” their earnings towards the end of their turn.
    5.1. The gamble button can be pressed once during each turn and can be applied to as many turns as the player wants.
    5.2. When the gamble button is chosen, the player can choose how much of their earnings to risk.
    5.3. If the player wins at the gamble button, their earnings are added and multiplied by a determined and static multiplier of 50%, but their chances of getting a favorable multiplier decrease.
    5.4. If they lose, they are set to lose the amount that they originally chose to risk.
    5.5. The result of the risk button will be randomized, but it will be skewed based on prior results of the risk button.
6. The player can choose to invest their earnings.

6.1. After choosing to invest, the player will be redirected to an investment screen that will display 5 different stocks.
6.2. The investment screen will be updated after each turn, displaying different sets of stocks every time.
6.3. Each stock will have a pop-up menu, showing whether to buy or sell. The pop-up will display a certain cap on how many stocks can be bought at once if the player chooses to buy.
6.4. The player will not gain the investment on that stock until the start of the next turn.

7. Players will have random events before the end of each turn.
   7.1. Random events will only occur when the player clicks the next turn and can cause the player to either lose money or gain money.
8. The goal at the end of the product will be to have acquired 1 million “dollars”.
   8.1. The term “dollars” is used loosely, as the unit of measurement is relative to the plot of the game.
9. The final product will be tested to ensure safe operation using unit tests.
   9.1. The software components of the game will be thoroughly tested through unit testing to ensure functions such as calculating the final score or profit are properly functioning.
   9.2. Different options – such as edge cases and unavailable options – will be explored to ensure that the player stays on a linear path towards the goal of the game.

4 Modeling Requirements

4.1 Use Cases

The use cases presented will be based on the Use Case Diagram created to show the overall pathway of the game. This will capture the high-level requirements of the project, as well as the overall general pathway the player is expected to take during gameplay.

Use Case: Oval

Actor: Stick Figure

Association: Solid line connecting actor to use case

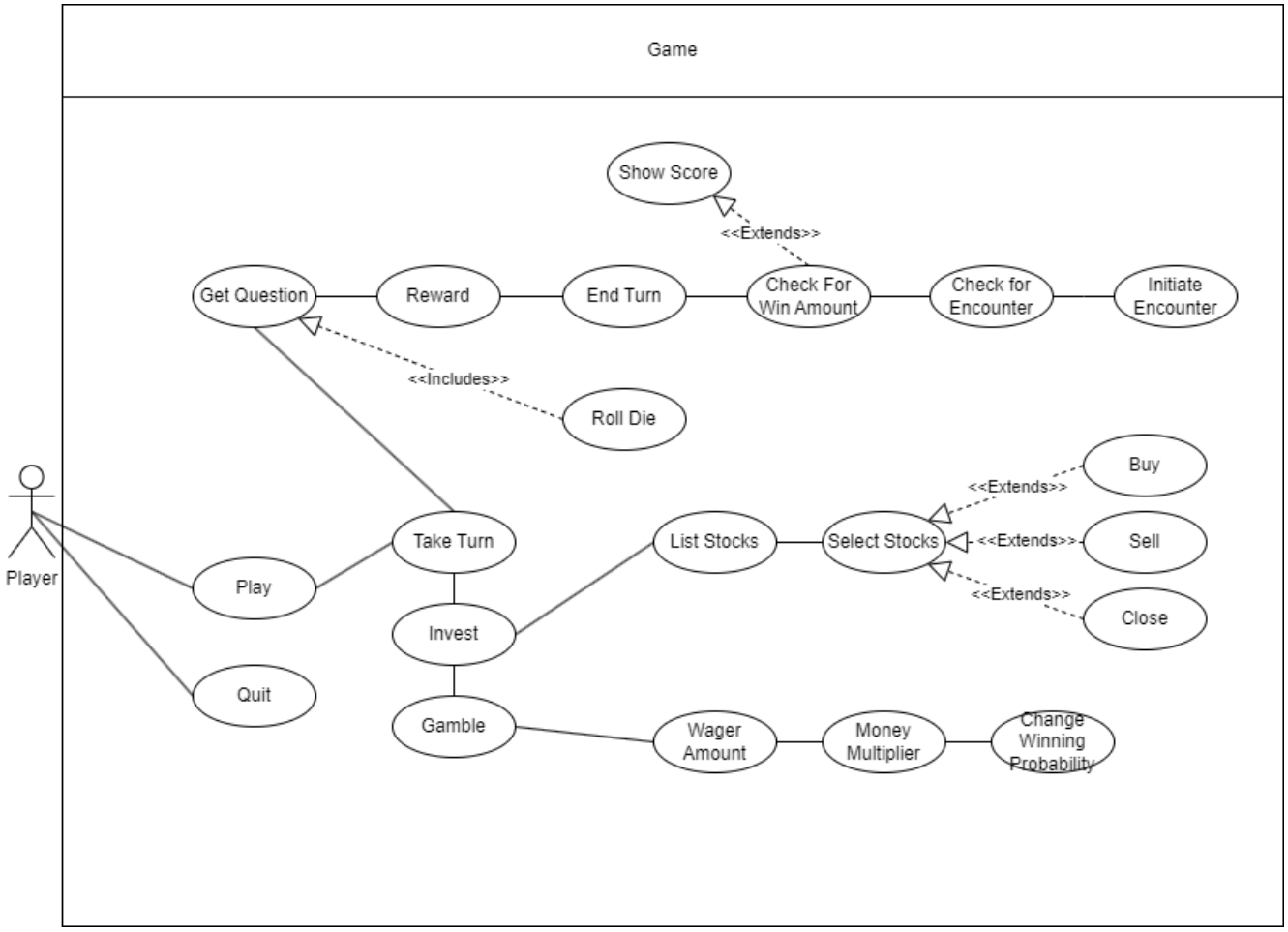


Figure 4.1.1: A use case diagram made for the program Daily Gambit

The use case presented will be the high-level requirements and pathway of the user experience. After starting the application, the player will be prompted with an option, 'play' or 'quit'. Play will lead to the main dashboard which is presented as three sub-sections. The player can progress the turn, invest, or gamble money that he/she has currently. Invest and Gamble will have its extended menus with their subsequent options. On the invest menu, stocks will be listed and will be able to be bought and sold. Users may close the menu to gamble or start their turn. Gambling is as simple as pressing the button and entering a wager amount, this then will be calculated with either a negative or positive multiplier depending on the current winning probabilities set by the game. These probabilities will be updated after every gambling event. To take their turn, the player will hit the take turn button, and be prompted a question based on a dice roll. After reaping the rewards or lack thereof, the user will move onto a random encounter which will have positive or negative effects on the user's current money, and enter the dashboard once again. This all repeats until the player wins by reaching the goal amount of money, thereby winning the game.

Use Case Name:	Play
Actors:	Player
Description:	The button you click to start the game.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Take Turn

Use Case Name:	Take Turn
Actors:	Player
Description:	The button clicked which proceeds the player to the next turn
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Roll Die, Investments, Gamble, End Turn

Use Case Name:	Roll Die
Actors:	Player
Description:	Rolls the dice (6 sided) to get the dice number
Type:	Secondary
Includes:	Get Question
Extends:	N/A

Cross-refs:	N/A
Uses cases:	N/A

Use Case Name:	Get Question
Actors:	Player
Description:	Gets the question depending on the dice roll number
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Reward, Take Turn

Use Case Name:	Reward
Actors:	Player
Description:	Gives money to the user
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Get Question, End Turn

Use Case Name:	Check for Encounter
Actors:	Player
Description:	Checks for if an encounter will happen on this turn

Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Check for Win Amount, Initiate Encounter

Use Case Name:	Initiate Encounter
Actors:	Player
Description:	Initiates the Random Encounter if there is one
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Check for Encounter

Use Case Name:	Invest
Actors:	Player
Description:	Opens the investment screen and allows the player to engage in investments
Type:	Primary
Includes:	List Stocks
Extends:	N/A
Cross-refs:	N/A
Uses cases:	List Stocks

Use Case Name:	List Stocks
Actors:	Player
Description:	Lists all the available stocks to buy and sell
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	List Stocks, Select Stocks

Use Case Name:	Select Stocks
Actors:	Player
Description:	Allows you to select the stocks you want to buy or sell
Type:	Secondary
Includes:	N/A
Extends:	Sell, Buy, Close
Cross-refs:	N/A
Uses cases:	N/A

Use Case Name:	Sell
Actors:	Player
Description:	Allows the user to sell a stock that they own
Type:	Secondary
Includes:	N/A
Extends:	Select Stocks

Cross-refs:	N/A
Uses cases:	N/A

Use Case Name:	Buy
Actors:	Player
Description:	Allows the player to buy a stock if they have enough money to
Type:	Secondary
Includes:	N/A
Extends:	Select Stocks
Cross-refs:	N/A
Uses cases:	N/A

Use Case Name:	Close
Actors:	Player
Description:	Closes the investment menu
Type:	Secondary
Includes:	N/A
Extends:	Select Stocks
Cross-refs:	N/A
Uses cases:	N/A

Use Case Name:	Gamble
Actors:	Player
Description:	The button that allows players to gamble their money with a chance of getting a big payout

Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Wager Amount

Use Case Name:	Wager Amount
Actors:	Player
Description:	The amount of money that is going to be wager is determined
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Money Multiplier

Use Case Name:	Money Multiplier
Actors:	Player
Description:	The amount of money the player gains multiplied by their current amount of money
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	Change Winning Probability

Use Case Name:	Change Winning Probability
Actors:	Player
Description:	Changes winning probability based on a win or lose
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	N/A
Uses cases:	N/A

Use Case Name:	Show Score
Actors:	Player
Description:	Shows the score the player has
Type:	Secondary
Includes:	N/A
Extends:	Check for Win Amount
Cross-refs:	N/A
Uses cases:	N/A

4.2 Class Diagram

The class diagram is used to identify the overall major objects and their relationships with each other.

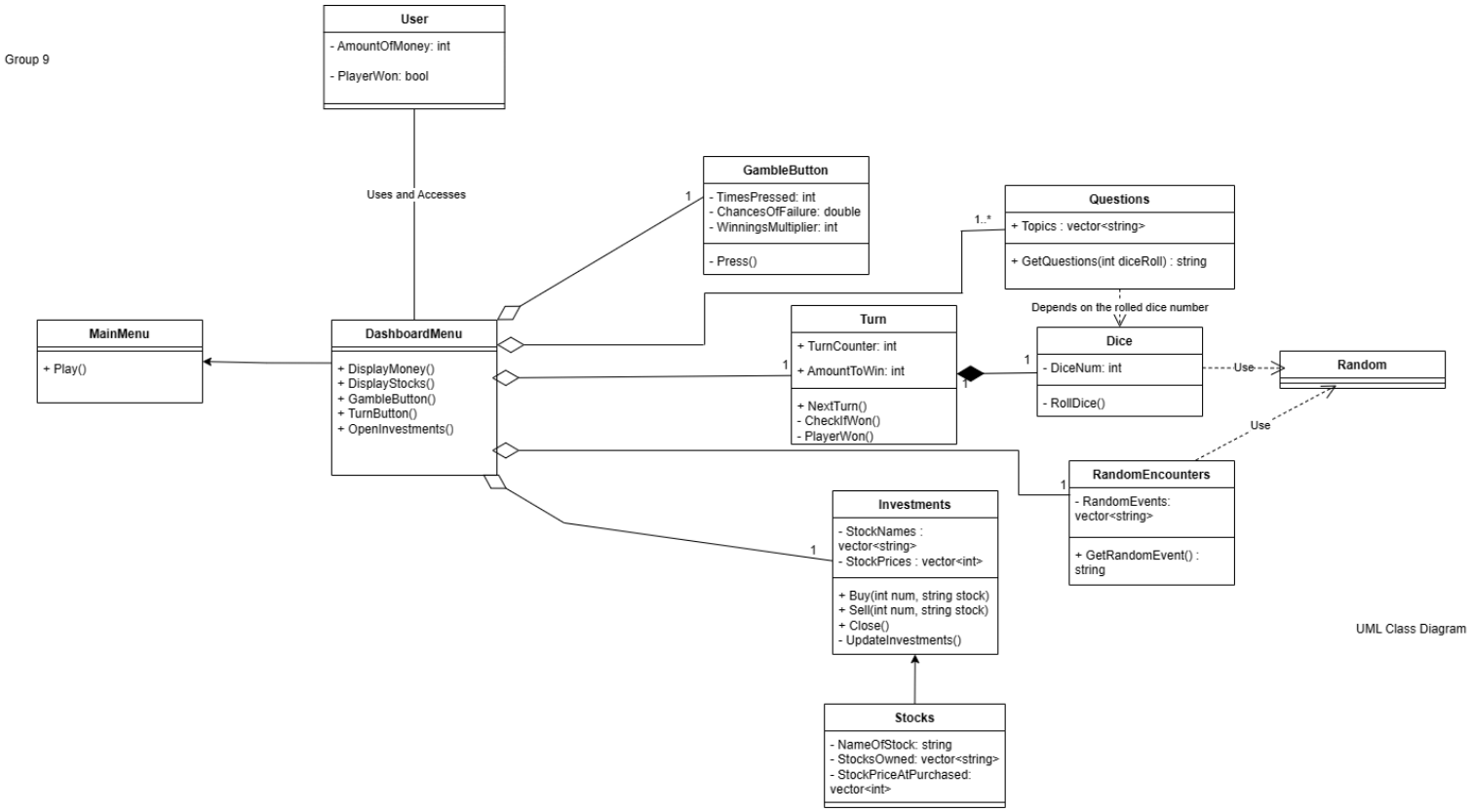


Figure 4.2.1: A class diagram made for the program Daily Gambit

Element Name		Description
DashboardMenu		The menu that will be presented to the player during each turn and used to play the game.
Attributes		
	N/A	
Operations		
	DisplayMoney(): void	A method that allows this class to look at the integer Money in the User class.
	DisplayStocks(): void	A method that allows this class to look at the vector of strings Stocks in the User class.
	GambleButton(): void	Activates the Press() function in the GambleButton class.
	TurnButton(): void	Activates the NextTurn() function in the Turn class.
	OpenInvestments(): void	Opens the GUI for the Investments class.
Relationships	User: Displays to DashboardMenu MainMenu: Brings you to the DashboardMenu GambleButton: Is a part of DashboardMenu Questions: Is a part of DashboardMenu Turn: Is a part of DashboardMenu Investments: Is a part of DashboardMenu RandomEncounters: Is a part of DashboardMenu	
UML Extensions	Uses and accesses User element Associations: GambleButton, Turn, Questions, RandomEncounter, Investments	

Element Name		Description
Dice		A die that is rolled after every turn to determine what question the user will get.
Attributes		
	DiceRoll: int	The dice roll of a six sided die.
Operations		
	RollDice(): void	Used to calculate the DiceRoll
Relationships	Turn: Uses Dice Questions: Depends on DiceRoll	

	Random: Used to find DiceRoll RandomEncounters: Depends on DiceRoll
UML Extensions	Includes: Random, Turn Extends: Questions

Element Name		Description
GambleButton		The button you can click to gamble. Win big or go home!
Attributes		
	TimesPressed: int	Tracks how many times the button has been clicked.
	ChancesOfFailure: double	Tracks how likely the user is to fail at the gamble button.
	WinningsMultiplier: float	Tracks how much the user can win in addition to their current money (For example 0.2 would be a 20% multiplier applied to the user's money if they won).
Operations		
	Press(): void	The method that activates every time the button is pressed. Used to calculate TimesPressed, ChancesOfFailure, and WinningsMultiplier.
Relationships	DashboardMenu: This is used in the DashboardMenu	
UML Extensions	Includes: DashboardMenu	

Element Name		Description
Investments		Stores the stocks and all their values every turn, this is also where the stocks can be bought.
Attributes		
	StockNames: vector<string>	The names of all the stocks.
	StockPrices: vector<int>	The prices of all the stocks, these two vectors are linked by having indices be the same (For example index 1 has the stocks name and its price).
Operations		

	Buy(int num, string stock): void	The user can buy the stock for the current price if they have the money, and add the stock name to the user variable StocksOwned.
	Sell(int num, string stock): void	The user can sell the stock for the current price, removing the stock name from user variable StocksOwned.
	Close(): void	Closes the investment screen.
	UpdateInvestments(): void	This is used in the next turn to update the stock prices after each turn.
Relationships	DashboardMenu: Is used in DashboardMenu Stocks: Stores stock names and values	
UML Extensions	Includes: DashboardMenu Extends: Stocks	

Element Name		Description
MainMenu		This element will contain the starting choices for the player before they enter the game.
Attributes		
	N/A	
Operations		
	Play(): void	This option will allow the player to choose to start and play the game.
Relationships	DashboardMenu: MainMenu is used to get to DashboardMenu	
UML Extensions	Extends: DashboardMenu	

Element Name		Description
Questions		Contains all the questions that need to be asked to the user after every turn.
Attributes		
	Topics: vector<string>	The questions themselves exist in this vector.
Operations		

	GetQuestions(int diceRoll): string	The questions are received based off of the number that the user gets in the dice roll.
	AnswerQuestion(): void	Displays and opens an input form to answer the displayed question once TurnButton() and NextTurn() is activated.
Relationships	DashboardMenu: Used in DashboardMenu	
UML Extensions	Depends on the result of RollDice() from Dice Includes: DashboardMenu	

Element Name		Description
Random		Used to calculate the DiceRoll.
Attributes		
	N/A	
Operations		
	N/A	
Relationships	Dice: Random is used to calculate DiceRoll	
UML Extensions	Used by Dice and RandomEncounters	

Element Name		Description
RandomEncounters		RandomEvents that may occur after any turn, these could be good or bad events.
Attributes		
	RandomEvents: vector<string>	The event text that would pop up if you got a random event.
Operations		
	GetRandomEvent(int diceRoll): string	This will use the Random class to get a RandomEvent that can cause the player to either gain or lose money.
Relationships	Dice: DashboardMenu:	
UML Extensions	Uses Random Includes: DashboardMenu	

Element Name		Description
Stocks		Stores all the stocks and their values.
Attributes		
	NameOfStock: string	Records the names of the stocks.
	StocksOwned: vector<string>	Records the stocks that the user owns.
	StockPriceAtPurchase: vector<int>	Records the price of the stock when the user bought it.
Operations		
	N/A	
Relationships	Investments: Uses Stocks	
UML Extensions	Includes: Investments	

Element Name		Description
Turn		The class that activates every time you want to go to the next turn, initiates classes that respond to turn.
Attributes		
	TurnCounter: int	Keep track of how many turns it has been.
	AmountToWin: int	Keep the number of money to exceed to win the game.
Operations		
	NextTurn(): void	Goes to the next turn and activates all the classes that need to be used to start the next turn.
	CheckifWon(): void	Checks if the player has the right amount of money to win after every turn.
	PlayerWon(): void	Sets PlayerWon boolean in User to true if the player has the right amount of money.
Relationships	DashboardMenu: Composition (Turn makes up DashboardMenu) Dice: Is a part of Turn (Turn uses dice to continue on)	
UML Extensions	Includes: DashboardMenu Association: Dice	

Element Name		Description
User		The scope of this class will pertain to the player of the game and what their “player stats” are as they progress through the game.
Attributes		
	AmountOfMoney: int	This will keep track of the earnings of the players.
	PlayerWon: int	This will be a condition that will become true when the condition to win the game is met.
Operations		
	N/A	
Relationships	MainMenu: Uses Stocks: Uses DashboardMenu: Uses and Accesses	
UML Extensions	Uses and Accesses DashboardMenu	

4.3 Representative Scenarios of System

4.3.1 Playing the Turn

The player initiates the game by clicking the "Play" button, triggering the "Start Turn" use case. This, in turn, activates the "Roll Die" use case, resulting in the generation of a random number. The system then proceeds to the "Get Question" use case, fetching a question corresponding to the dice number. After the player answers the question, the flow diverges based on correctness. A correct answer invokes the "Gain Money" use case. The sequence continues, guiding the player through the game dynamics.

4.3.2 Investing in Stocks

Upon clicking the "Investments" button, the player engages in the "Investments" use case, which invokes the "List Stocks" use case. The system displays available stocks, allowing the player to select stocks for buying or selling in the "Select Stocks" use case. This may further lead to the execution of the "Buy" or "Sell" use cases. Upon completing these transactions, the player clicks "Close," triggering the "Close" use case and concluding the stock investment process.

4.3.3 Gambling

The player decides to gamble by clicking the "Gamble" button, initiating the "Gamble" use case. This leads to the "Wager Money" use case, where the player determines the amount of money to wager. The outcome of the gamble may result in either winning, triggering the "Gain Money Multiplier" use case, or losing, leading to the "Lose Wagered Money" use case. Subsequently, the "End Turn" use case is activated, marking the conclusion of the gambling event and initiating other turn-related events.

4.4 Sequence Diagram

Lifeline: Vertical dashed line representing the lifespan of an object

Message: Arrowed line indicating communication between objects

Gambling

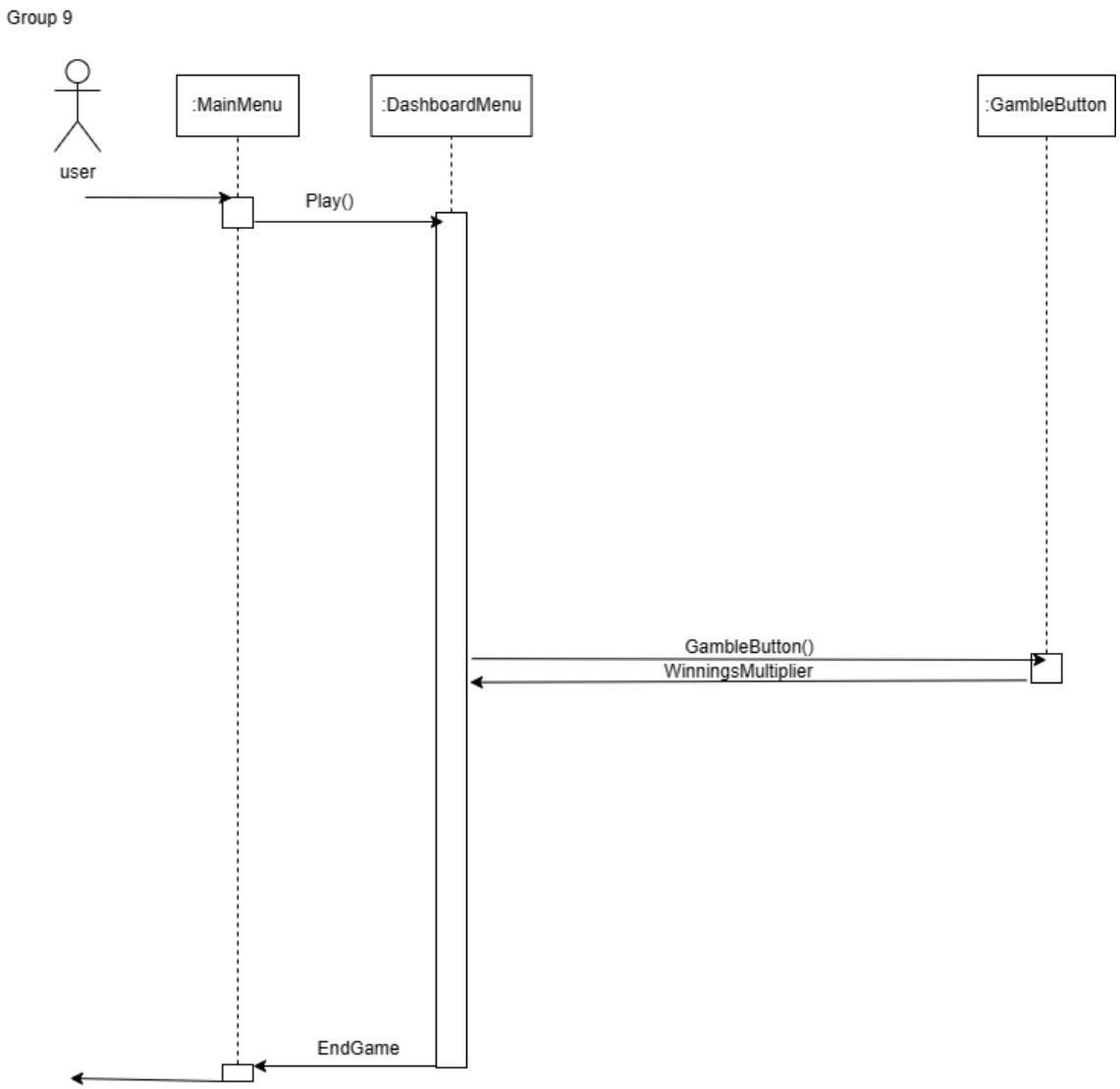


Figure 4.4.1: A sequence diagram made for the program Daily Gambit. The user in this diagram wants to press the gamble button

Investing in Stocks

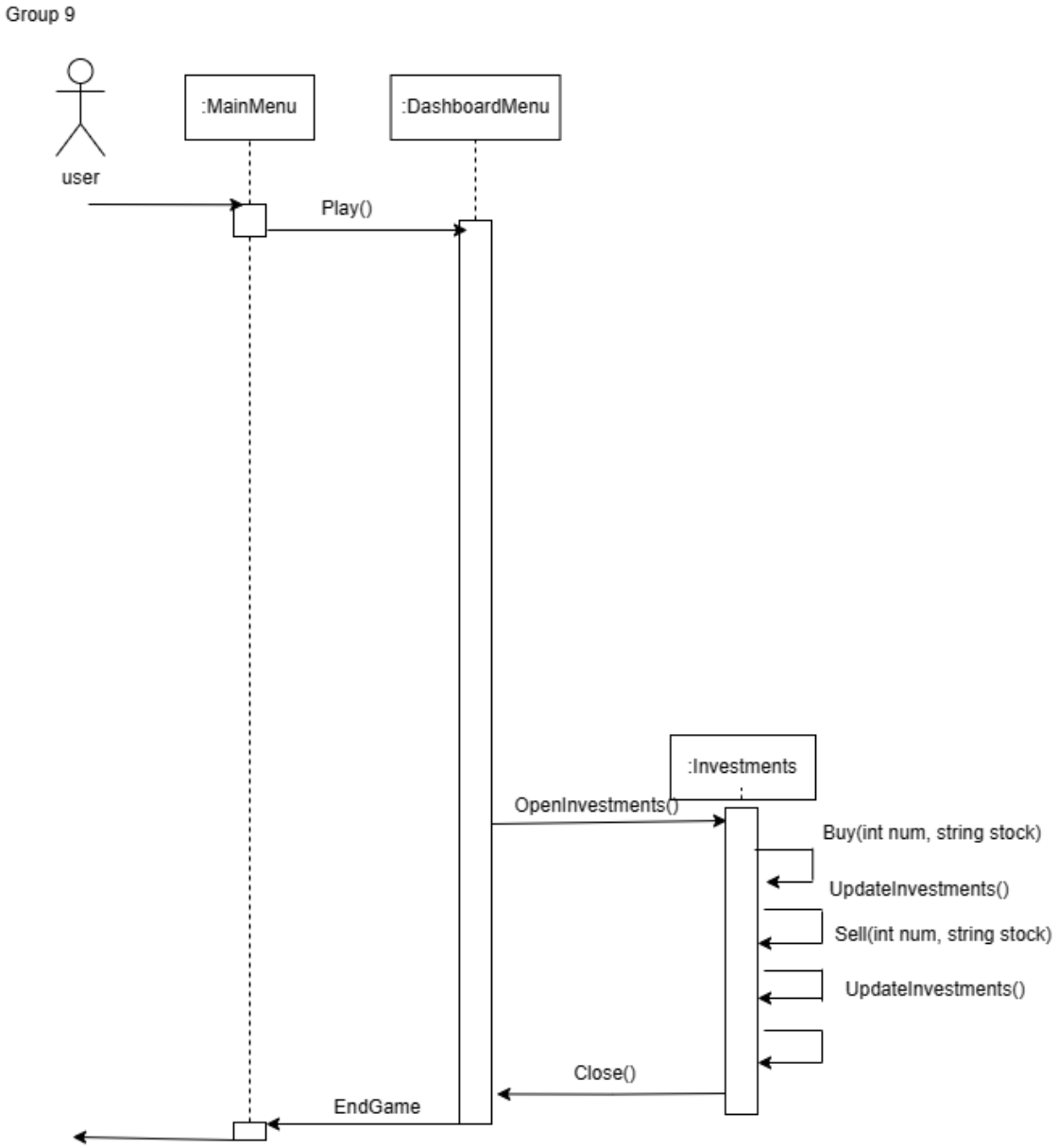


Figure 4.4.2: A sequence diagram made for the program Daily Gambit. The user in this diagram is investing in stocks

Playing with Turn

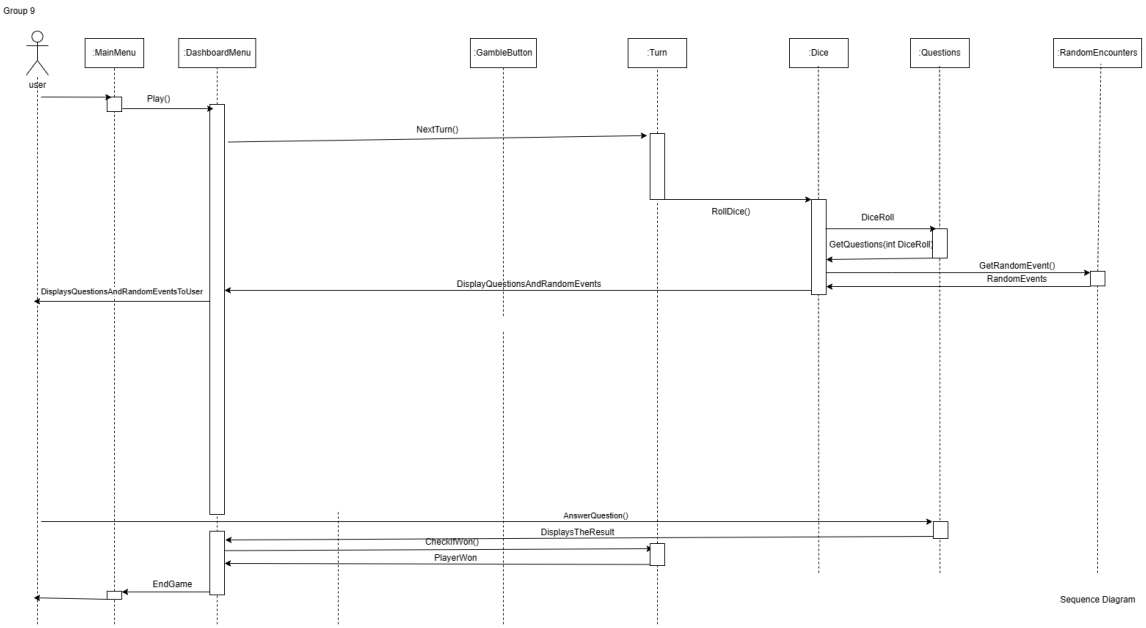


Figure 4.4.3: A sequence diagram made for the program Daily Gambit. The user in this diagram is playing the game turn-by-turn

4.5 State Diagram

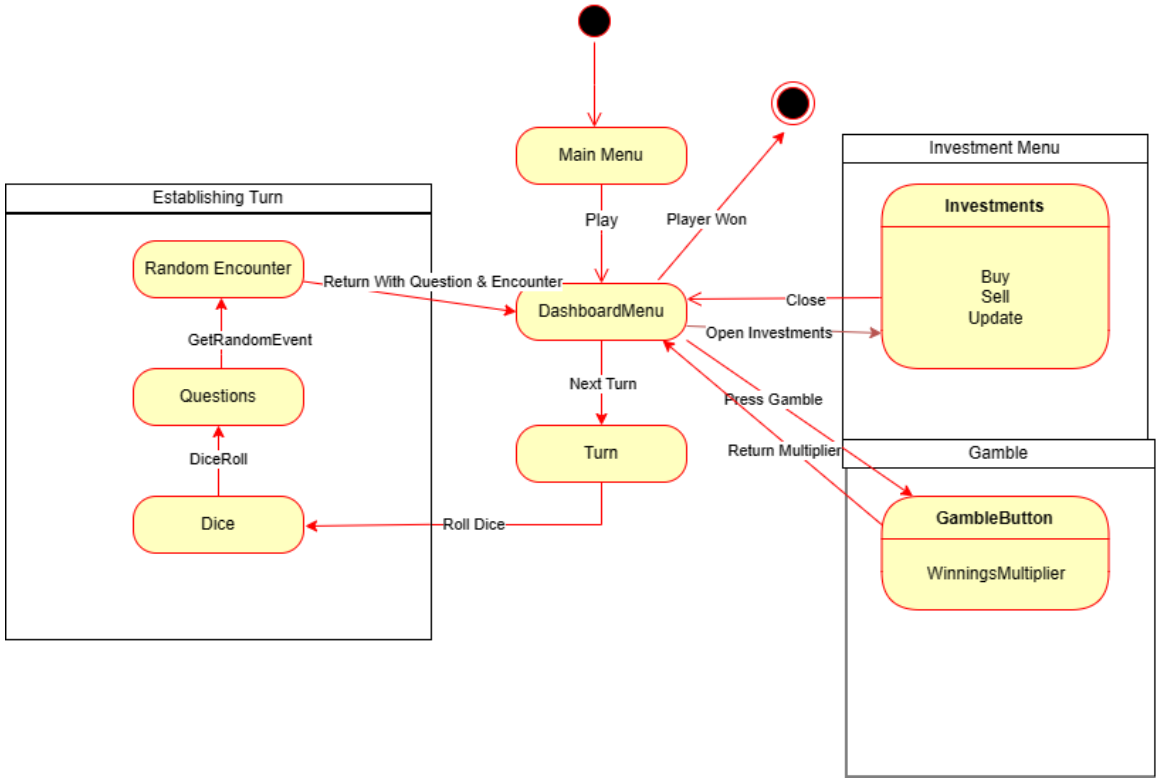


Figure 4.5.1: State diagram of the Daily Gambit program

The state diagram demonstrates general scenarios to describe the behavior of when a player plays the game and uses the interaction functionalities of the UI.

5 Prototype

The Daily Gambit prototype [5], also known as DG for short, will demonstrate a dynamic and interactive user interface featuring responsive forms and buttons. The journey begins in the main menu, where players are introduced to the overall goal of the game, stating that to complete the game, the user must complete a series of questions in order to reach the milestone of having a million dollars.

In addition to the task and goal of the game, the main menu also features a prominent "Start Game" button that invites users to initiate gameplay. Upon selecting "Start Game," users are seamlessly redirected to the game screen, where the dashboard controls take place. The dashboard prominently displays the current amount of money held by the player and provides information about the stocks in their possession. Interactive buttons further enhance the user's engagement by offering access to different game features.

The investment button unlocks an interactive investment menu, empowering users to make informed decisions about buying or selling stocks in which they will have to select which stocks are available within the store or in their inventory to purchase and sell respectively. Once purchased or sold, a notification will appear on the screen to confirm that the user bought or sold stocks respectively, and overall, this system not only contributes to the player's net worth but also adds a strategic dimension to the game. The gamble button introduces an element of risk, allowing users to wager their current money for a chance to make a potential profit. If users successfully wager their current money, the success rate decreases each time, otherwise, the player loses the money they wagered for and the success rate will remain the same. Overall, the success rate and winnings multiplier provide crucial information, and outcomes are immediately reflected in the player's financial status.

Another significant feature is the start question button, triggering an educational challenge for users. The questions generated are mathematical themed and topics will be randomly picked based on the dice’s roll. So for instance, if the die rolls on a 6, the mathematical topic will involve exponential problems and equations. The questions will be displayed on the center top of the user’s device screen and the input form will be shown on the center middle along with the submit button for users to submit their final answer. Once submitted, the game will display whether or not the user answers correctly or not. Random events are also another feature of the game that can sometimes occur to negatively or positively affect the player by giving additional money or losing more money and is tied together when users click on the start question button. The integration of an instant feedback mechanism ensures that users receive timely information about the outcomes of their actions, whether it be investment choices, gambling results, or the correctness of their answers.

The overall design prioritizes a user-friendly interface, with responsive forms and buttons contributing to a seamless and enjoyable experience. Smooth transitions between different screens and components enhance the overall flow of the game. The combination of financial decision-making, risk-taking through gambling, and educational challenges

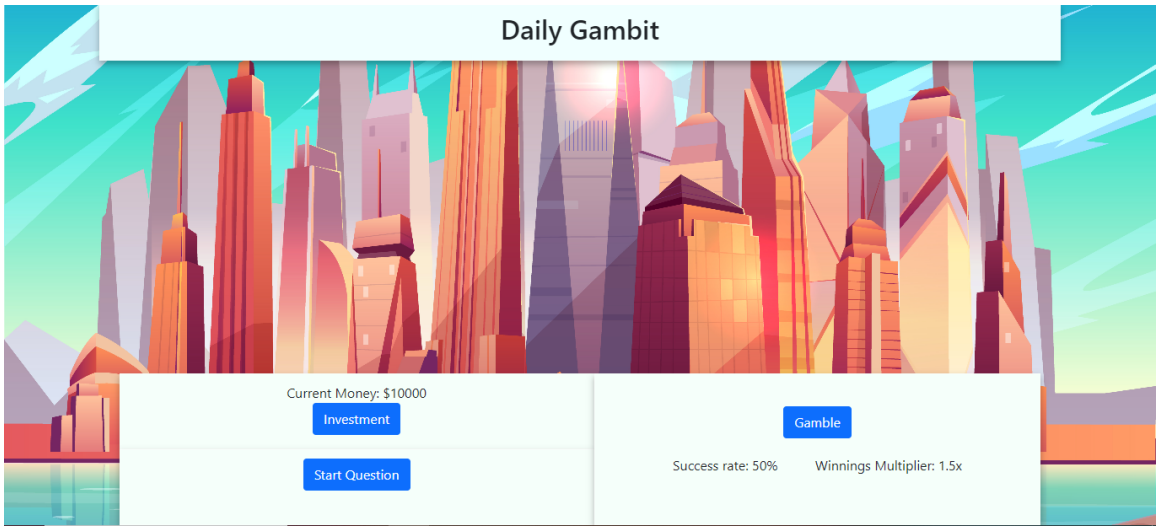
creates an immersive and dynamic gaming environment, ensuring that users remain engaged and entertained throughout their DG experience.

5.1 How to Run Prototype

The first version of the prototype, Daily Gambit, will serve as a visual representation of the envisioned application. It is primarily accessible through a web interface, requiring a modern web browser for optimal viewing such as Google Chrome, Microsoft Edge, and Mozilla Firefox. Since it requires modern web browsers to view and use the game, a stable internet connection will be required which is one of the constraints of the prototype along with the need for a web browser to access it. Users can explore the provided interfaces to understand the design elements and functionalities and there are no specific system configuration or plugin requirements in order to use or access the application. The prototype will be accessible through this URL link: https://daily-gambit-game.netlify.app/

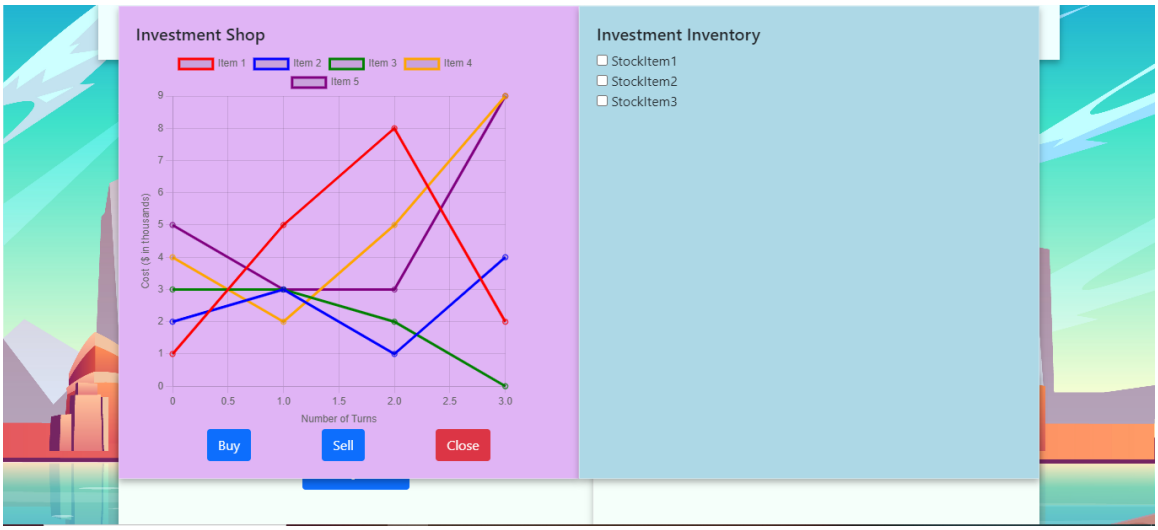
However, to run the prototype locally, a modern web browser like Google Chrome is needed to download if you do not have one. An IDE or integrated development environment also needs to be downloaded in order to view, edit, and run the files. Download Visual Studio Code and once it successfully downloads, download or clone the project repository of the prototype where it currently resides on GitHub. This can typically be done using a version control system like Git through the command git clone project URL within the terminal in your IDE like Visual Studio Code, or by downloading a ZIP archive of the project inside one of your computer's directories. Once this is done, find the directory in which you downloaded or cloned or if you are already in the directory of the prototype, find the index.html file of the prototype and click run in Visual Studio Code and click start debugging. After doing so, you will have the option to select which debugger you would like to run the application such as Microsoft Edge or Google Chrome. Once you click one of them, you will be able to run the prototype without issue.

5.2 Sample Scenarios



The game has 3 different sections, Investment, Question, and Gamble that the user can interact with to earn rewards.

5.2.1 Investment

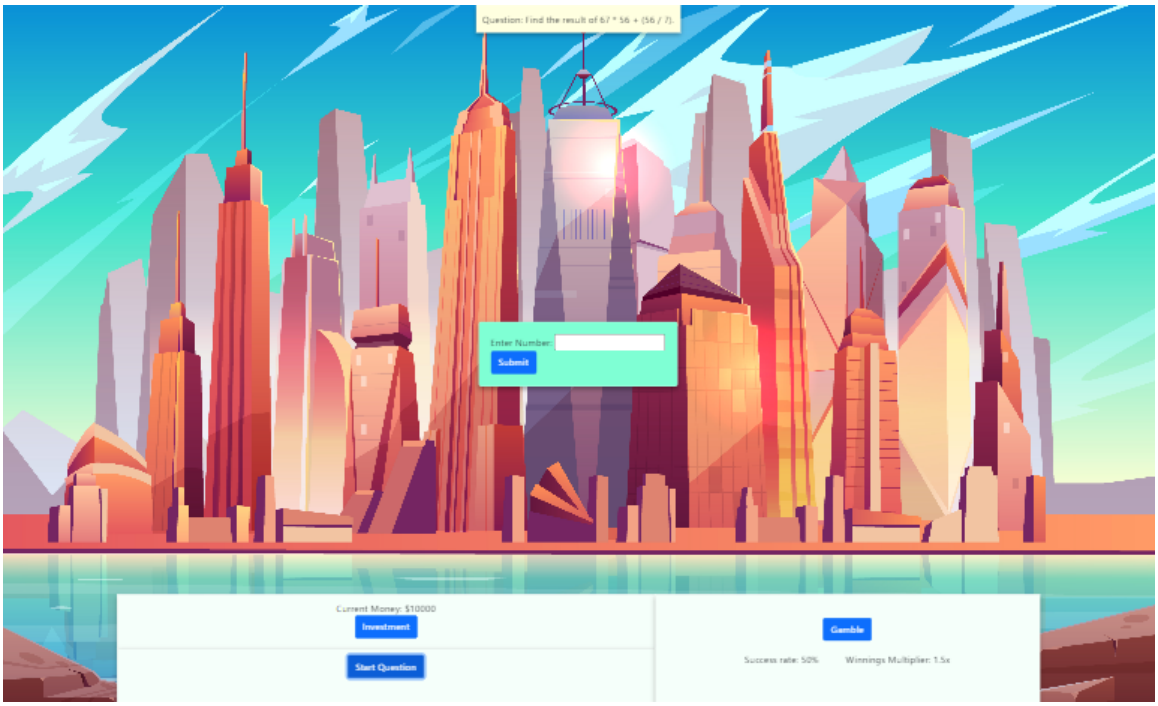


Investment window allows users to use their “Current Money” to buy stocks that are available on the market.

The Investment window has 2 subsections, the Market section and the Inventory section. The Market section shows what stocks are currently available on the market. The Inventory section shows what stocks are bought and owned by the user.

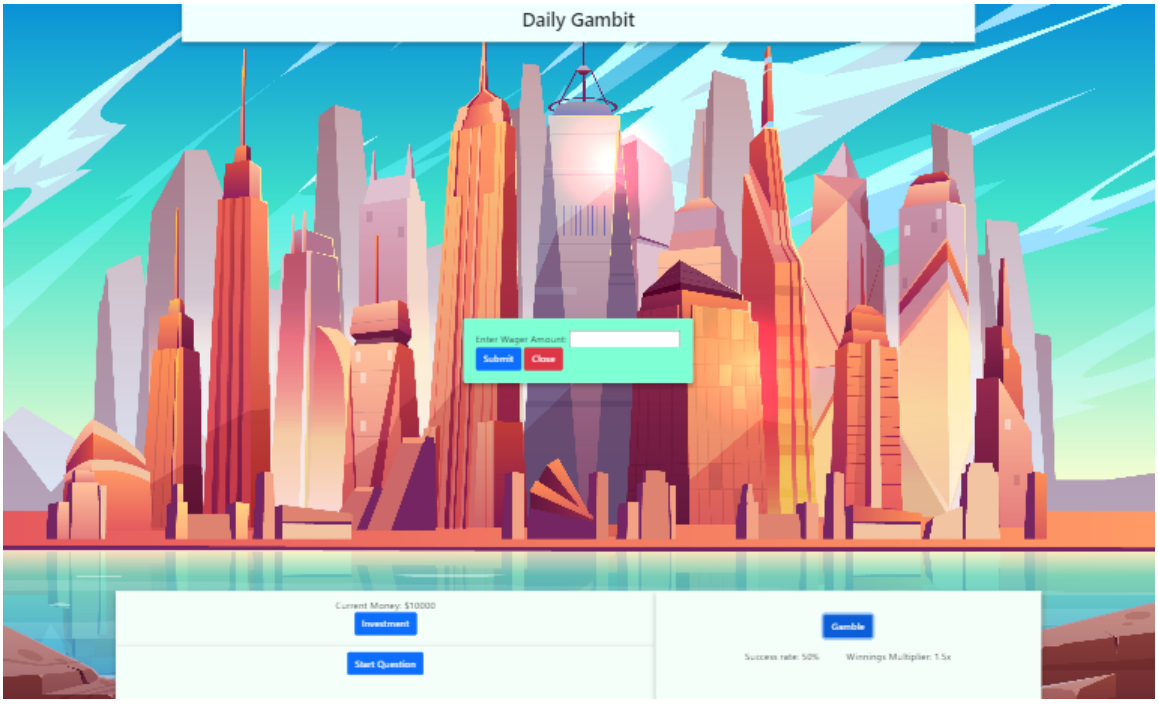
The user can freely choose to hold on to their stocks or sell them to earn or lose money.

5.2.2 Question



Question section is where users can test their knowledge. It generates a wide range of math problems, from solving exponential equations to basic algebra. Each correct answer to the question rewards the user money which they can spend to purchase stocks.

5.2.3 Gamble



The Gamble button allows users to double-down their risk and reward on their next question encounter.

6 References

Website entry: *https://daily-gambit-game.netlify.app/*

[1] Massachusetts Curriculum Framework. (2017). *MATHEMATICS Grades Pre-Kindergarten to 12*. MATHEMATICS Grades Pre-Kindergarten to 12 . https://www.doe.mass.edu/frameworks/math/2017-06.pdf

[2] *Draw.io - free flowchart maker and diagrams online*. Flowchart Maker & Online Diagram Software. (n.d.). https://app.diagrams.net/

[3] *UML sequence diagram online tool*. SequenceDiagram.org - UML Sequence Diagram Online Tool. (n.d.). https://sequencediagram.org/

[4] Tea teatime. (n.d.). *Teateatime/daily_gambit*. GitHub. https://github.com/teateatime/daily_gambit

[5] Microsoft. (2021, November 3). *Visual studio code - code editing. redefined*. RSS. https://code.visualstudio.com/

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.