Assignment 2

Team number: 20
Team members:

Name	Student Nr.	Email
Andrés Lot Camarena	2724030	a.lotcamarena@student.vu.nl
Stefan Dragosh Vatamanu	2726157	s.vatamanu@student.vu.nl
Marcel Niedzielski	2726454	m.niedzielski@student.vu.nl
Simona Peychinova	2724076	s.peychinova@student.vu.nl

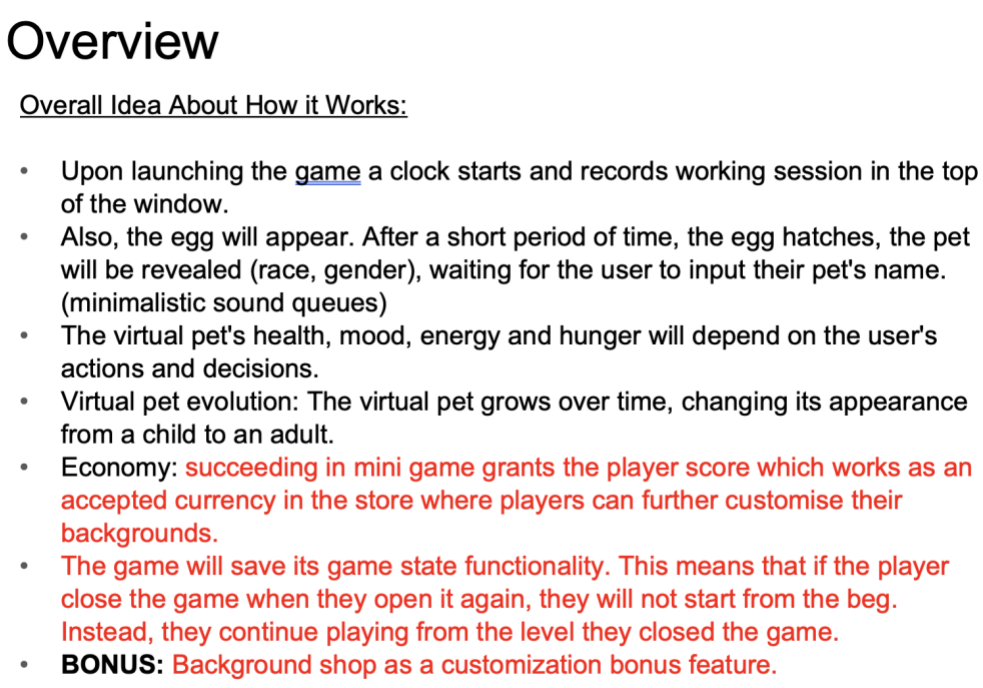
Do NOT describe the obvious in the report (e.g., we know what a name or id attribute means), focus more on the **key design decisions** and their "**why**", the pros and cons of possible **alternative designs**, etc.

Format:
Class
objects
attributes
operations()
associations
ENUMERATIONS

Summary of changes from Assignment 1

All team members took part in this section.

The changes are visible in the pictures below:



Overview

Overall Idea About How it Works:

- Upon launching the game a clock starts and records working session in the top of the window.
- Also, the egg will appear. After a short period of time, the egg hatches, the pet will be revealed (race, gender), waiting for the user to input their pet's name. (minimalistic sound queues)
- The virtual pet's health, mood, energy and hunger will depend on the user's actions and decisions.
- Virtual pet evolution: The virtual pet grows over time, changing its appearance from a child to an adult.
- Economy: succeeding in mini game grants the player score which works as an accepted currency in the store where players can further customise their backgrounds.
- The game will save its game state functionality. This means that if the player close the game when they open it again, they will not start from the beg. Instead, they continue playing from the level they closed the game.
- **BONUS:** Background shop as a customization bonus feature.

Functional features

ID	Short name	Description	
F1	Buttons	The player will interact with different menus by pressing buttons containing icons indicating their corresponding menu. Buttons will be located in top and bottom bars surrounding the virtual pet. - Feed: Displays food options to buy and feed pet - Sleep: Puts the pet to sleep - Bath: Cleans pet - Play: Starts mini-game with pet - Stats: Displays hunger, energy, mood and health of pet - Shop: Displays customizable items to buy	
F2	Pet interactions	Players can feed, play, and put their pets to sleep by using buttons.	
F3	Mini-game	The player can boost their pet's "mood" by playing our mini game and gain money when they succ.	
F4	Aging and evolution	Pets will grow older over a certain time, evolve from "child" to "adult".	
F5	Death	If the player doesn't take care of their pet's needs, it can cause the pet to die and a gravestone takes its place (clock still runs).	
F6	Clock/stopwatch	Initiated when game is launched. Tracks the amount of time the app has been open.	

Here is a summary of the changes made to address feedback in Assignment 1:

- Reduced the number of mini games from more than one to a single game:
  This change was likely made to simplify the development process and ensure that the game will be well-designed. Focusing on a single mini game may be more effective than trying to create several mini games with varying levels of quality.

- Added a background shop as a customization bonus feature:
  This change provides users with an additional way to personalize their gameplay experience. By adding a shop where players can purchase new backgrounds, the game becomes more engaging and encourages users to spend more time with it.

- Implemented saving game state functionality:
  This change ensures that the users can save their process and start where they left the game, even if they close the game or shut down the device.

- Modified pet instructions to use buttons instead of cursor movements:
  It simplifies the gameplay mechanics and makes the game more intuitive.

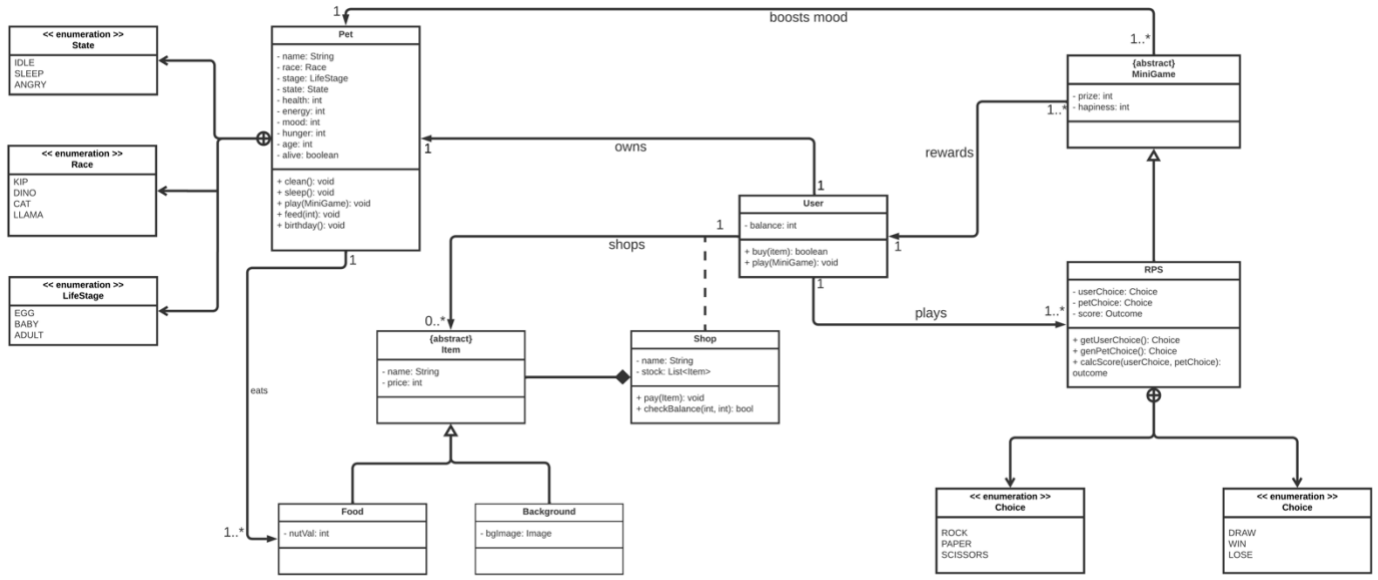
- Removed security requirements:
  This change is made to ensure that the game is not overly complicated for users.

- Added a feature where the pet earns money upon successful completion of mini games: This change encourages the users to play the mini game and adds sense of progression and reward to the gameplay experience. This could provide the user with a sense of accomplishment.

Class diagram

All team members took part in this section

This chapter contains the specification of the UML class diagram of our system, together with a textual description of all its elements.



Pet

The **Pet** class represents a Tamagotchi virtual pet with various attributes such as name, health, energy, etc. The most relevant attributes are state, race, and stage, as they are enumeration types that classify the pet's current phase. One of the most relevant operations is birthday(), which is triggered after a fixed amount of time adds up 1 to the age of the pet and boosts its mood as well. Other operations such as clean(), sleep(), play(), and feed() are used to interact with the pet and maintain its overall well-being.

State

The **State** enumeration represents the current state of the pet. It has three values: IDLE, SLEEP, and ANGRY.

Race

The **Race** enumeration represents the breed of the pet. It has five values: KIP, DINO, CAT, and LAMA.

LifeStage

The **LifeStage** enumeration represents the life stage of the pet. It has three values: EGG, BABY, and ADULT.

Item

The **Item** class is a superclass that represents the buyable items in the game. It has two attributes: name, and price.

The **Item** class has a composition relationship with the **Shop**, as the **Shop** class contains a list of **Item** objects.

Food

The **Food** class is a subclass of **Item,** which represents the food that the pet can eat. It has one unique attribute nutVal, which is an int representing the nutritional value of the food. The **Food** class also has one method, consume(), which is called when the user buys the food.

The class has a many-to-one association with the Pet class, as one pet can eats one or more foods.

Background

The **Background** class is a subclass of **Item** represents the background image of the game. It has one attribute, bgImage, which is an Image representing the background image of the game.

Shop

The **Shop** class is an association class that makes possible for a user to buy an item. It has an attribute called stock which is a list of Item objects that the shop has for sale. The **Shop** has a method called buy() which allows the user to purchase an item from the shop by passing the item as an argument. The **Shop** class relates to the **Item** class through composition, which means that the **Shop** is composed by the items that are for sale. The **User** class is also connected with the **Shop** class, as the user can buy items through a **Shop** object. Overall, the **Shop** class plays a critical role in the game, allowing the user to feed their pet and customize the environment.

MiniGame

The **MiniGame** class is an abstract class that represents mini-games that can be played in the game. It has two attributes, prize and happiness, which represent the prize awarded to the player upon winning the mini-game and the effect of the mini-game on the pet's mood. The class is related to the **User** and **Pet** classes through many-to-one associations, which represent the fact that one or more mini games can reward one user and boost the mood of one pet.

RPS

The **RPS** class represents a Rock-Paper-Scissors mini-game that can be played in the game. It has three attributes, userChoice, petChoice, and score, which represent the user's choice, the pet's choice, and the outcome of the game, respectively. The calcScore() operation is used to calculate the outcome of the game based on the user's and pet's choices.

The **RPS** class is associated with the **Choice** and **Outcome** enumerations, which represent the different options and outcomes of the game.

Choice

The **Choice** class is an enumeration type that represents the possible choices that a user or pet can make in the Rock-Paper-Scissors (RPS) mini-game. The choices are ROCK, PAPER, and SCISSORS. They are used as arguments to the RPS class's calcScore method, which calculates the outcome of the RPS game.

Outcome

The **Outcome** enumeration type represents the possible outcomes of the Rock-Paper-Scissors (RPS) mini-game regarding the user. The possible outcomes are W, D, and L.
This enumeration is used in calcScore method to represent the outcome of a game. The method takes two instances of the **Choice** enumeration as parameters. Based on these, the calcScore method returns an instance of the **Outcome** enumeration, representing the outcome of the game.

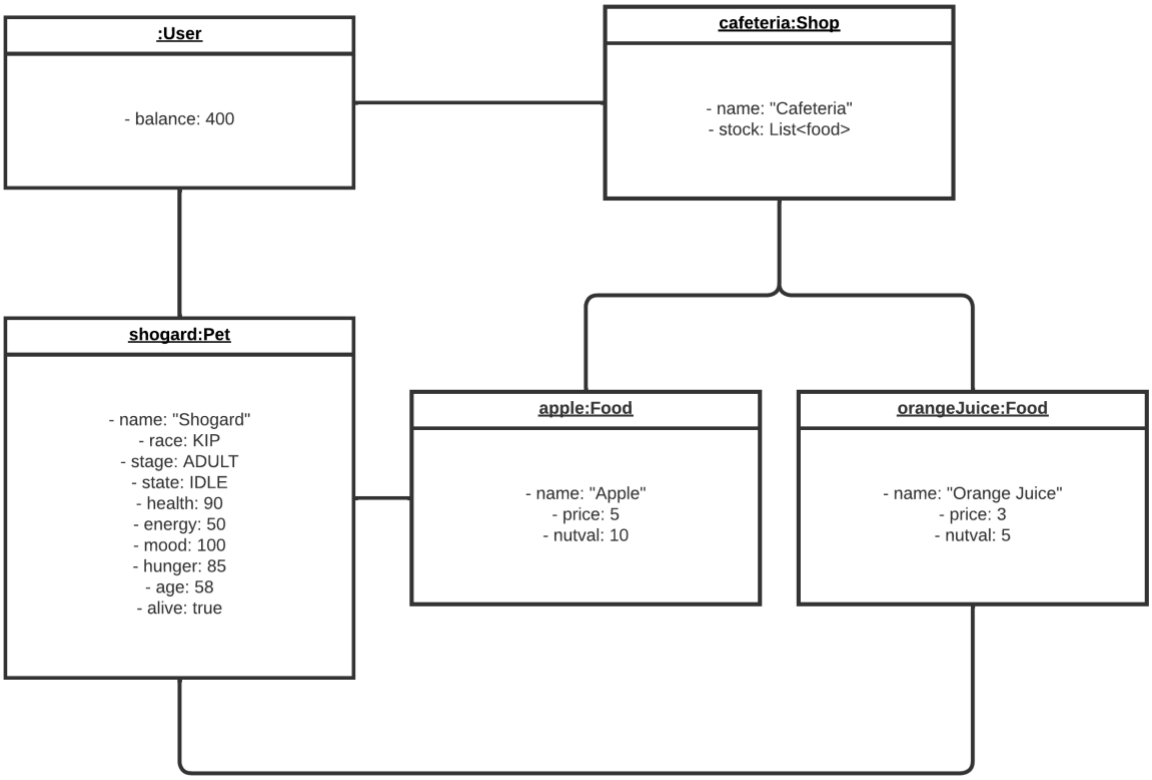
User

The **User** class represents the game user. It has one attribute - balance which represents the user's balance of in-game currency. The pay() operation is used to deduct the specified amount from the user's balance when buying thing from the shops. The class is related to **Pet** through a one-to-one association, which represents the fact that one user can own one pet.

Object diagram

Andres and Dragosh made the diagram for this section.

This chapter contains the description of a "snapshot" of the status of your system during its execution. This chapter is composed of a UML object diagram of our system, together with a textual description of its key elements.



The UML object diagram represents a snapshot of a moment in the Tamagotchi game where a *user* is buying **Food** objects from the *cafeteria* to feed() their *pet,* therefore filling its hunger attribute. The diagram is composed of four objects: *user, cafeteria, apple*, and *orangeJuice*, and one **Pet** object, *shogard.*

User has one attribute named balance which shows the amount of money the user owns for buying items. The *cafeteria* is an object of **Shop**, with two attributes: stock, which is a list of available food items, and name. The stock is represented by a list of **Food** objects, which in this case, includes *apple* and *orangeJuice*.

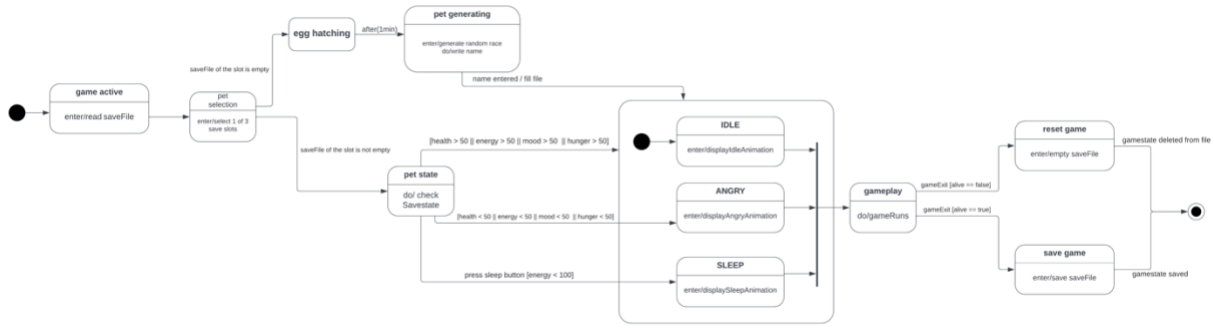
Apple and *orangeJuice* have three attributes: name, price and nutval. Nutval is the attribute that shows how much the hunger will be filled after the pet is fed up.
Finally, the **Pet** object, *shogard*, has numerous attributes including stage, state, health, alive, etc.

The diagram illustrates how the objects are connected. *User* owns the pet and is also connected to *cafeteria* with a line that indicates a purchase is taking place. The *cafeteria* is then connected to *apple* and *orangeJuice* to represent the items that are available for purchase. Finally, *Shogard* is connected to the **Food** objects with a line that represents the feeding process.

State machine diagrams

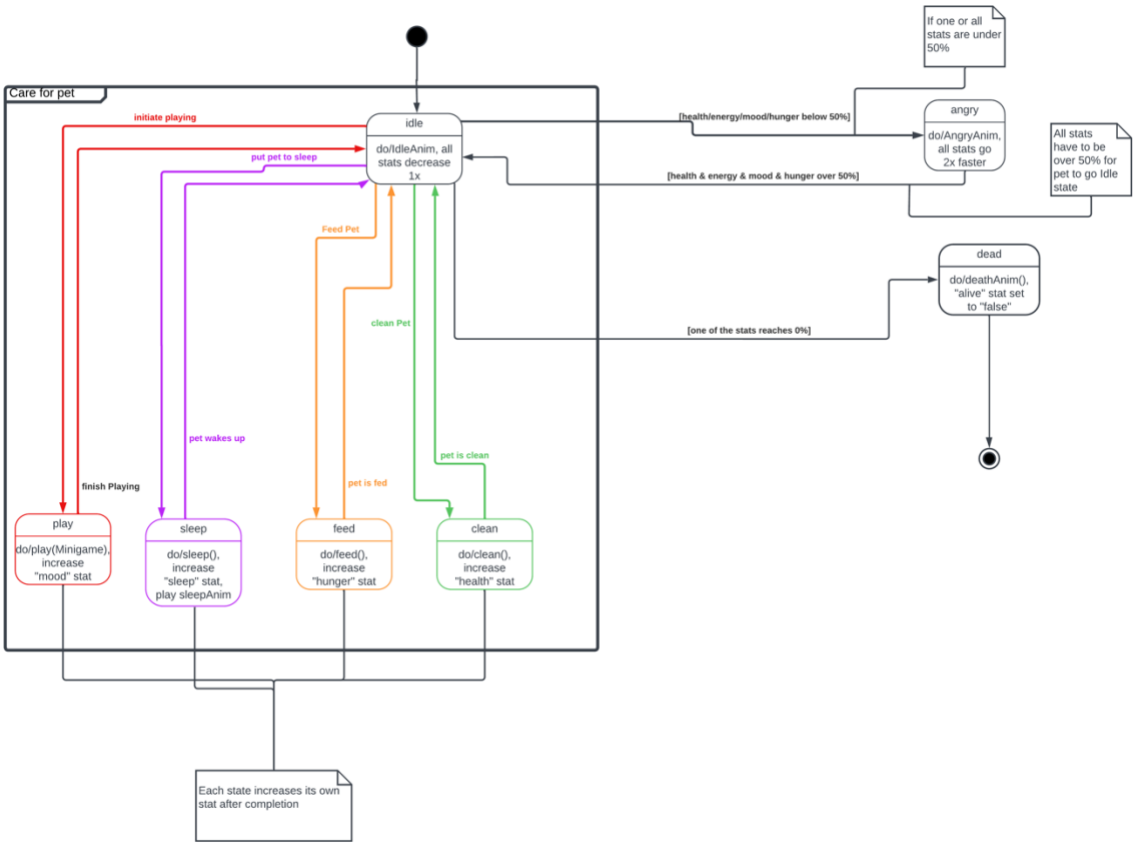
This chapter contains the specification of at 2 UML state machines of our system, together with a textual description of all their elements.

First state machine diagram: Saving game state
Simona and Marcel took part in this section.



This diagram is a representation of how loading and saving the game state would work. After starting the game, 3 save slots are displayed. After selecting one, it's saveFile is loaded, and checked for content. If it does not contain a saved state, that starts the process of creating a new pet. The egg is hatched, the race and sex are randomly generated, and the user is asked for the desired name of his pet. After the name is entered, the pet enters the idle state, and the actual gameplay starts. If the file contains a saved state after loading, the state of the pet in which the game was exited is checked and triggered, then the gameplay starts. If the user wishes to exit the game, the isAlive boolean is checked. If evaluates to false, the saveFile is emptied and the game exits. If isAlive evaluates to true, the saveFile is updated with current stats of the pet etc. After that the game is exited.

Second state machine diagram: Pet state
Dragosh took part in this section.



The Pet State diagram represents different states and transitions that will occur within the life of the pet. A pet has 5 states, idle, feed, sleep, play, and clean. By default the pet is in his idle state, where the “idleAnim” animation runs. In this state, every stat degrades at the normal rate. If any of the stats goes under 50%, the pet transitions to the angry state, where it will run the “angryAnim” animation, and each state will start degrading twice as fast. It remains in this state until all stats are above 50%.

The rest of the states are triggered by user actions. The pet will enter feed state when the player decides to feed the pet, sleep state when the pet is put to sleep, play state when the user plays with the pet and clean state when the user wants to clean the pet. After completion of each of these states, the respective stat will increase, and the pet transitions back to idle.

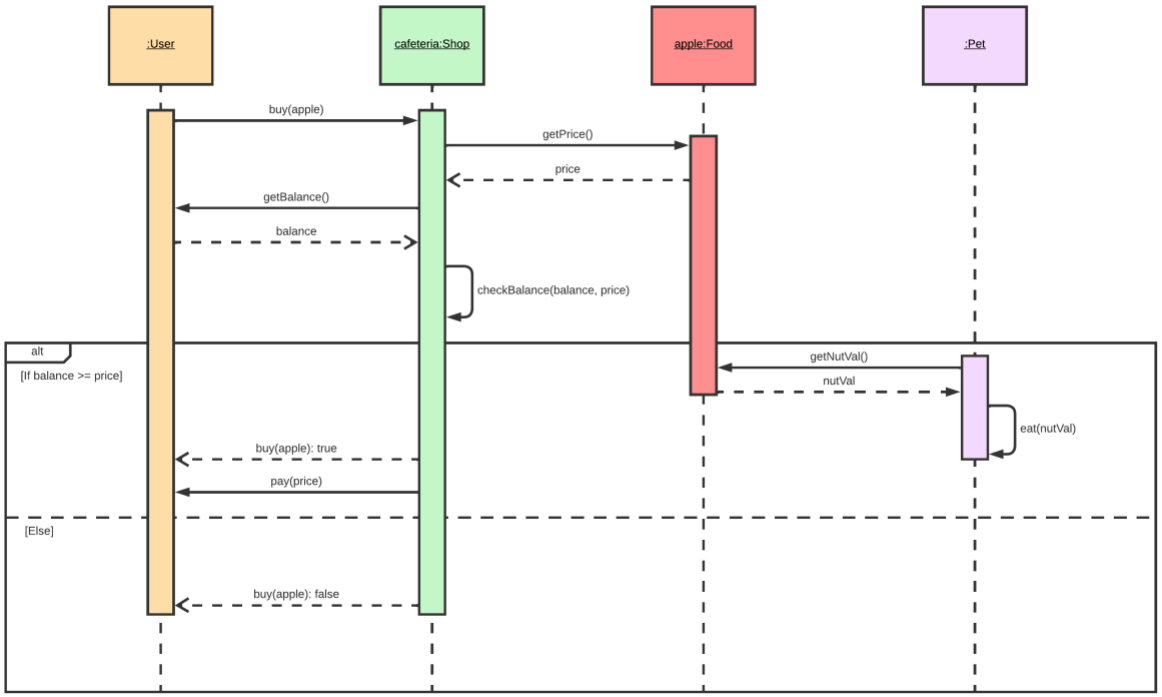
If any of the stats drops to 0%, the pet will die, transition into Dead State and the user is forced to start over with a new pet.

Sequence diagrams

Andres and Marcel made the diagrams for this section.

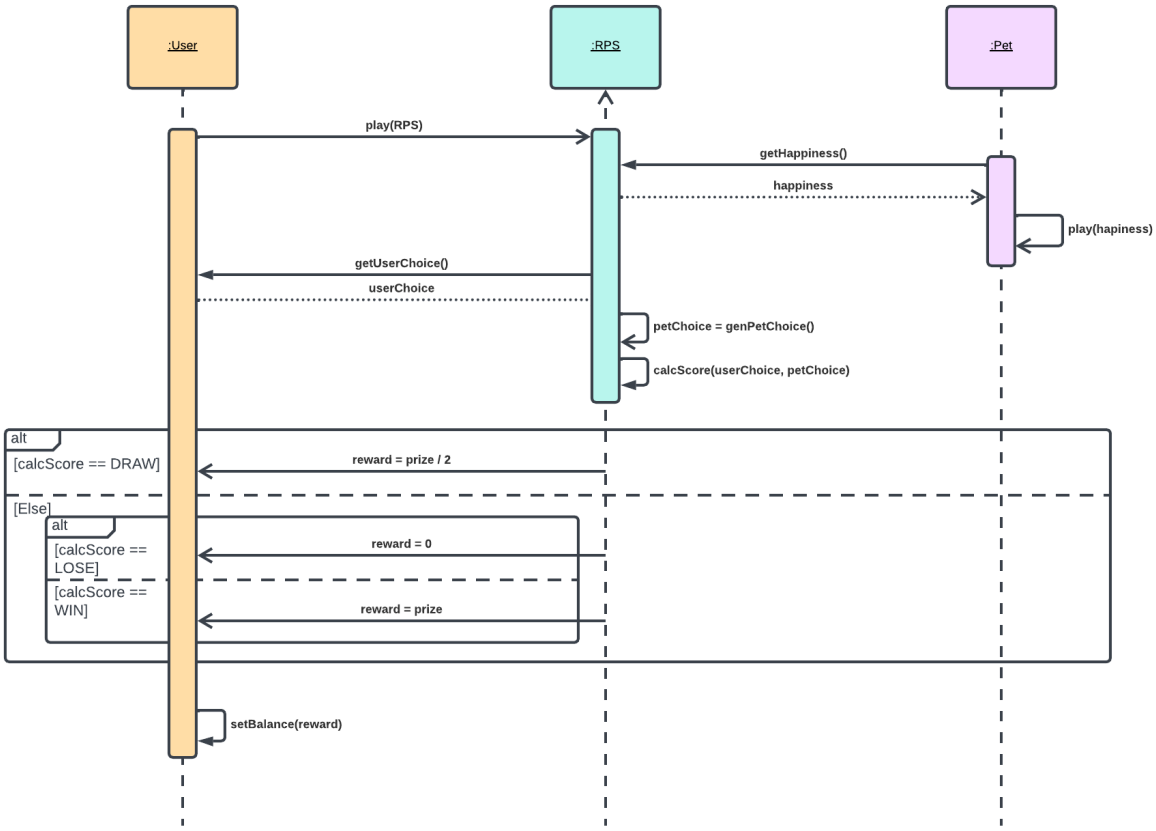
This chapter contains the specification of at 2 UML sequence diagrams of our system, together with a textual description of their elements. scenario, in a special case that may happen (e.g., an error situation), when finalizing a match, etc.

First sequence diagram: Process of buying food from cafeteria



This sequence diagram represents the process when the user buys food to feed the pet.
First, the user opens the *cafeteria*, chooses a product (in this case an *apple*), and wants to buy it. In. order to buy the apple the user should have enough balance. After this is checked, if the user has enough money to buy the *apple* the pet gets its nutritions which fill up its the hunger and user pays for the apple (decreasing balance). If the user does not have enough money, they cannot buy the apple, money is not withdrawn, and pet does not get any nutrition values.

Second sequence diagram: RPS game



This sequence diagram shows the process of playing a Rock-Paper-Scissors mini game with the pet. User decides to play the mini game and pet gets the happiness value, no matter the outcome of the game. Then RPS object asks for user input for the mini game and randomly generates one for pet. After having these 2 values it can calculate the score. Depending on the outcome of the match the user gets rewarded respectively. If it draws it will get half of the prize added to balance, if he loses, he would notget any money and if he wins, he will get the full amount.

Time logs

	A	B	C	D
1	**Member**	**Activity**	**Week number**	**Hours**
2	All team members	Class diagram	1	5
3	Simona	Description of class diagram	1	1
4	Dragosh and Lot	Object diagram	1	1
5	Simona and Marcel	First state machine diagram	1	2
6	Dragosh	Second state machine diagram	1	2
7	Lot and Marcel	First sequence diagram	1	3
8	Lot	Second sequence diagram	2	2
9	Simona	Description of object diagram	2	0,3
10	Dragosh	Description of second state machine diagram	2	0,3
11	Marcel	Description of first state machine diagram	2	0,3
12	Lot	Description of first sequence diagram	2	1
13	Simona	Description of second sequence diagram and format of document	2	1
14			2	
15			2	
16			2	
17				
18				
19				
20				
21			**TOTAL**	19,3