# IE 201 PROJECT PART – 3

**Group 3**

Samet Taha İstegün - 2019402081

Burak Berk Bulut – 2019402138

Batuhan Özkan - 2018402033

Sude Yılmaz - 2019402054

IE 201

Ali Tamer Ünal – Zeki Caner Taşkın

January 2022

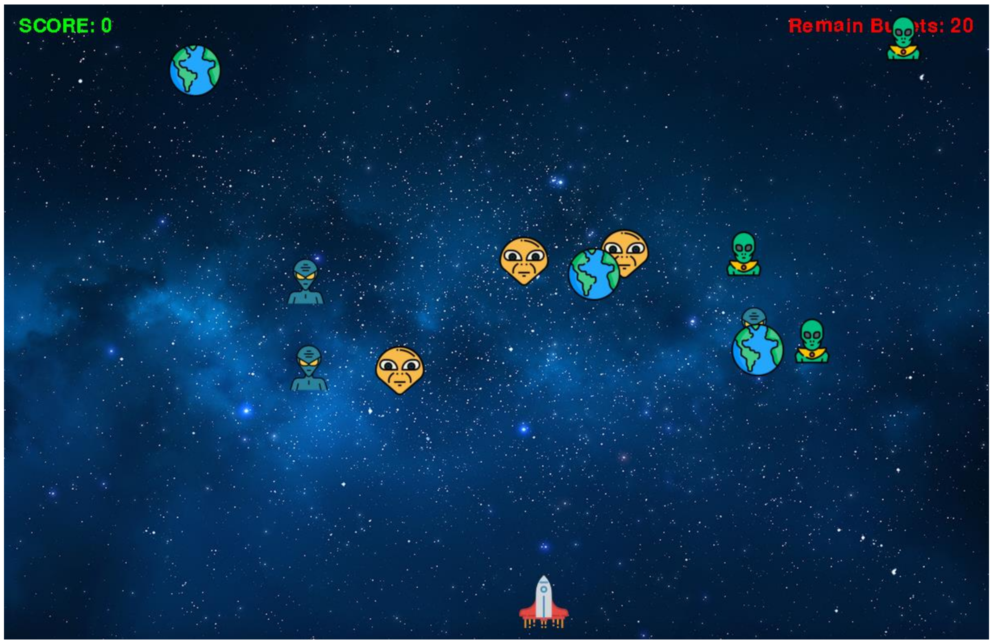
## Contents

## A Brief Description of The Game

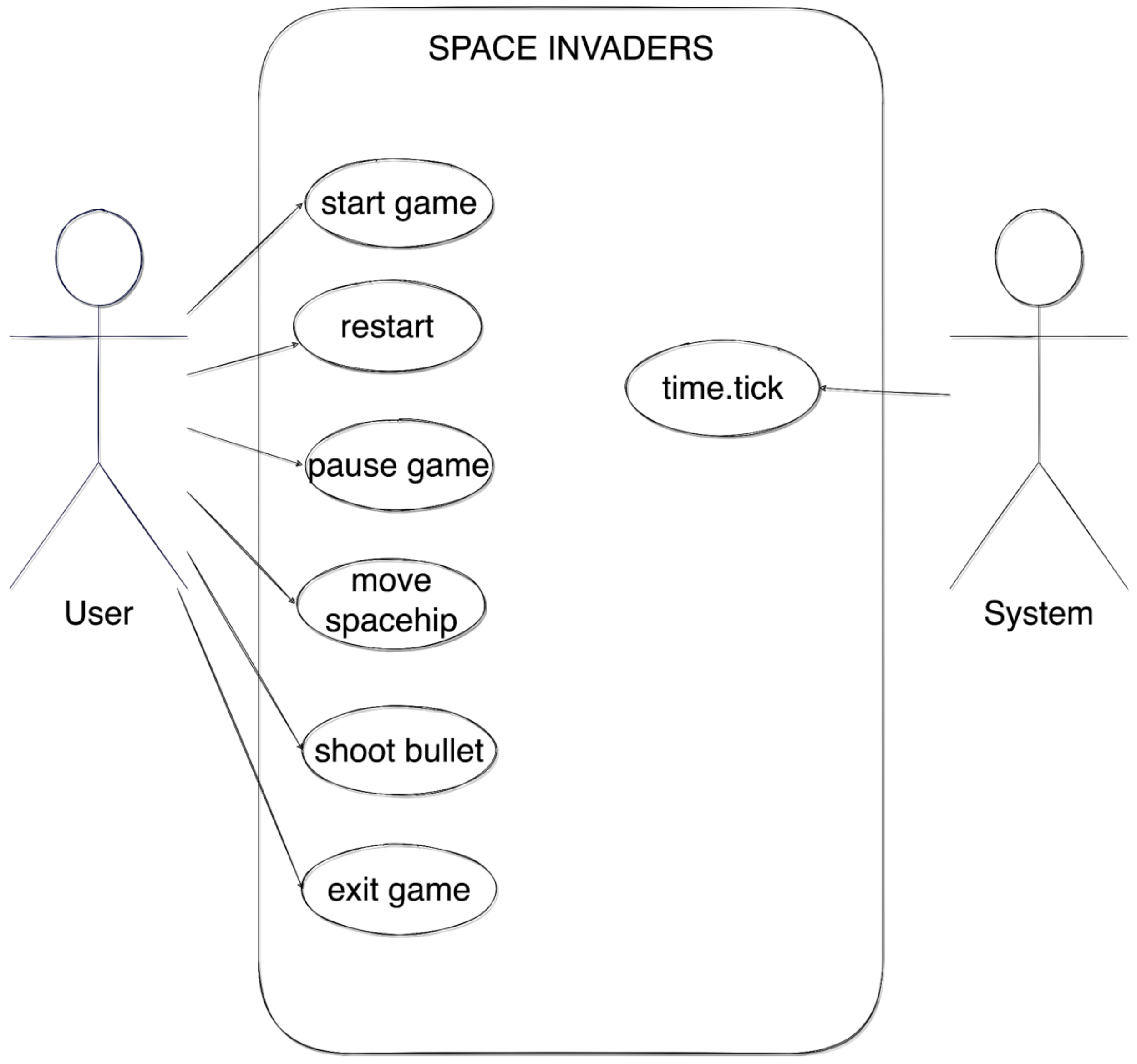
Space game is a  game that works with the logic of hitting randomly incoming targets with bullets by a user-controlled spaceship. The spaceship is the user object of the game that can move left and right on the floor of the game screen and shoot upwards. At the top of the screen, space objects appear in random places and are constantly moving. These space objects can be the following two types:

- **Enemies:** Targets that the user must hit to win.
- **Allies:** Trap targets that the user should not hit.

The user has zero score and a certain number of bullets at the beginning of the game. When the user hits an enemy object via the spaceship, both the user's score and the number of bullets he has are increased. However, when the user hits an ally object, both the score and number of bullets are decreased. If the user runs out of bullets the game ends. Below is a similar example of the game we designed:

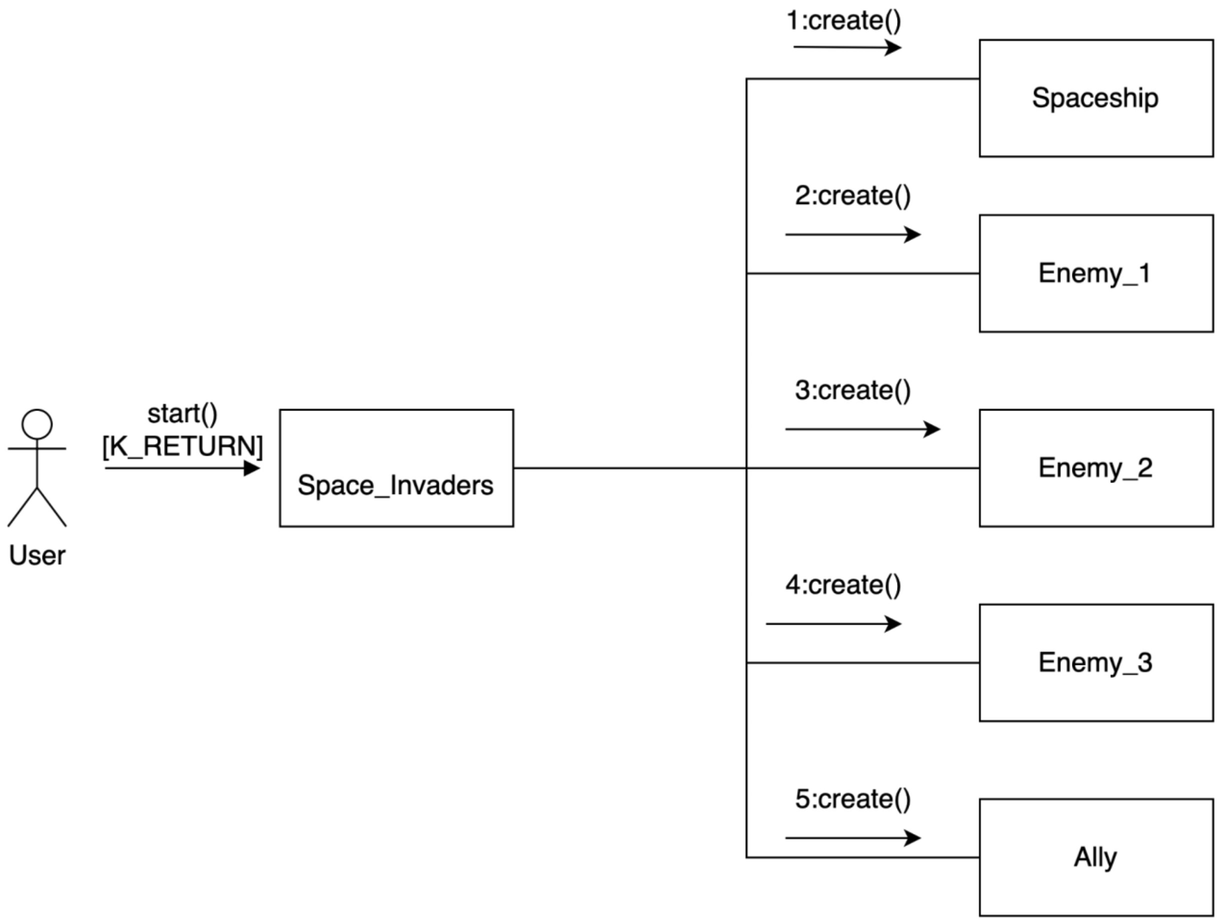


## Use Case Diagram



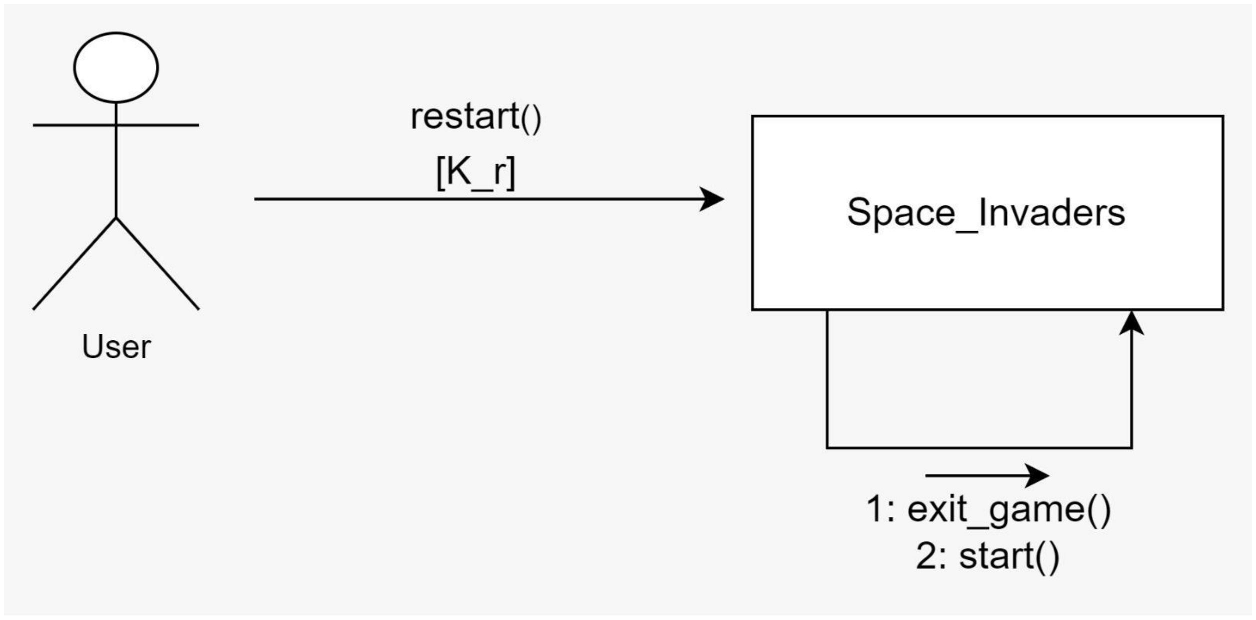
## Collaboration Diagrams

### 1. start



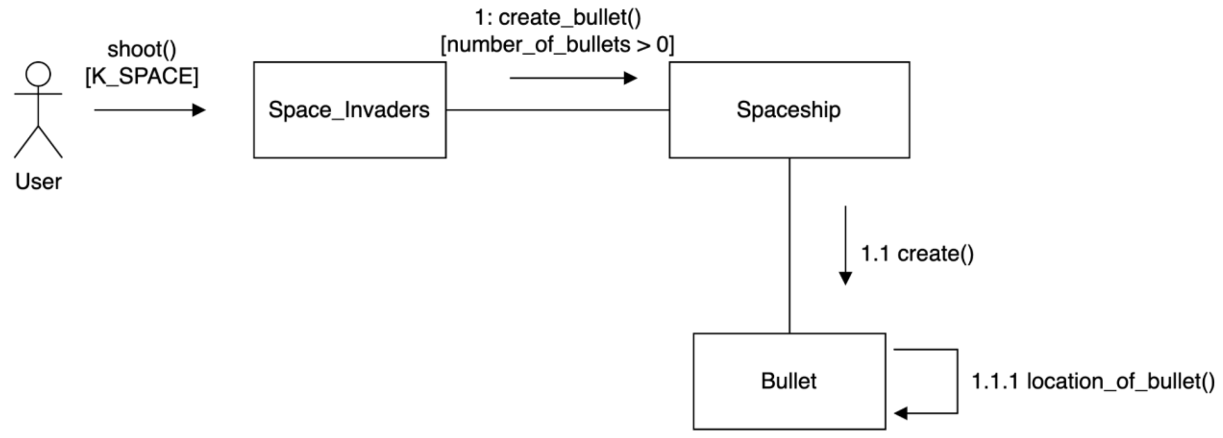
"start()" use-case creates all game objects(Enemy, Ally, Spaceship) that main game class has.

## 2. restart



This use-case, first, deletes all game objects(Enemy, Ally, Spaceship) that Space_Inavders has with the "exit_game()" message and gets the default value of the score and number of bullets. Then, new game objects(Enemy, Ally, Spaceship) are created with the "start()" message.

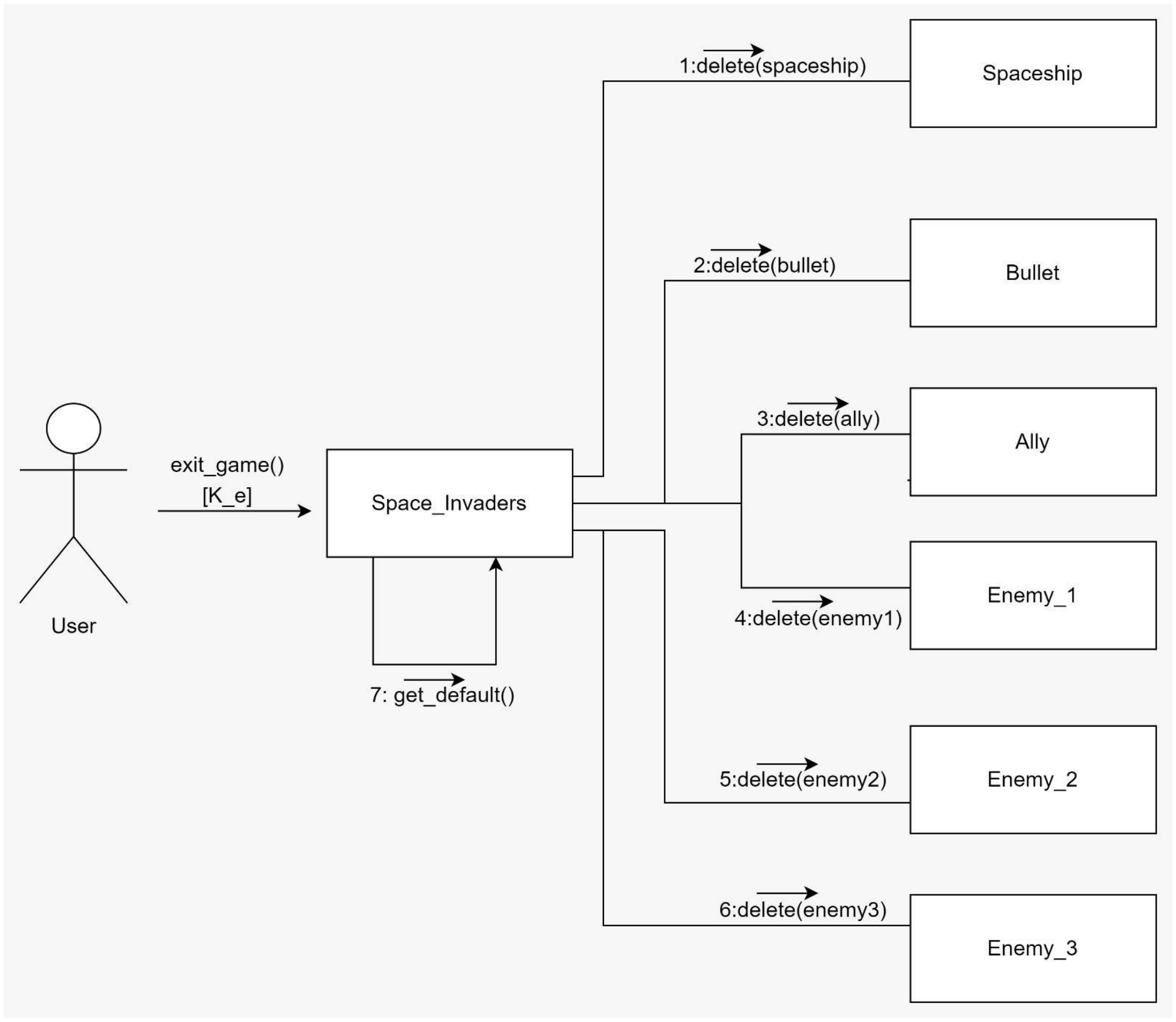
## 3. shoot



Initially,  this use case checks how many bullets we have. If the number of bullets is greater than zero, class Space_Invaders sends a message to class Spaceship via calling the create_bullet() function.

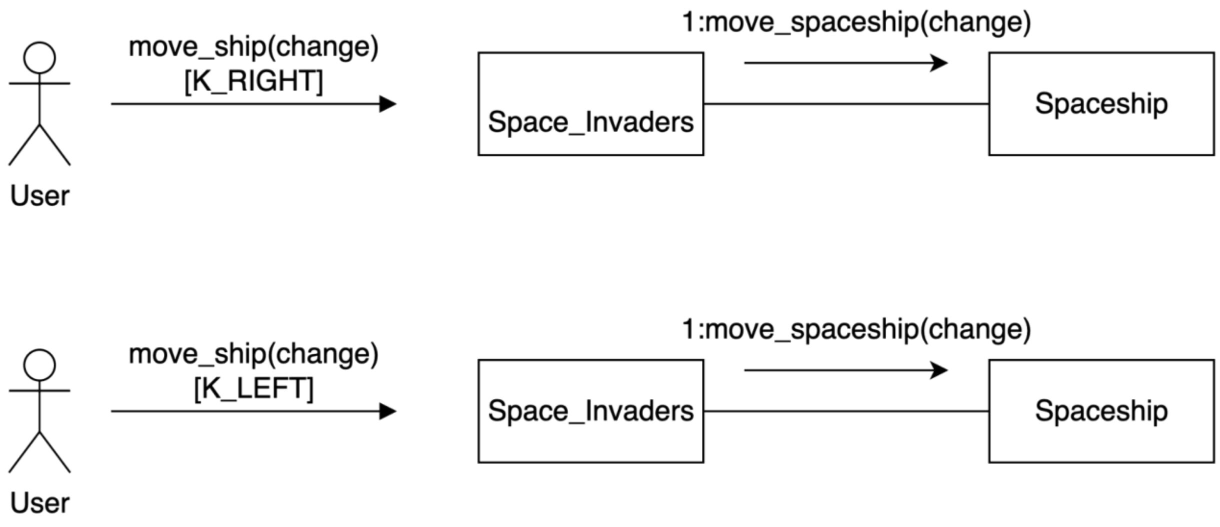
Then the Spaceship class sends create() message to the Bullet class to create a bullet object. Lastly, the bullet object calls itself with the location_of_bullet() function to determine its location that it is created with respect to the instant location of the spaceship.

## 4. exit_game



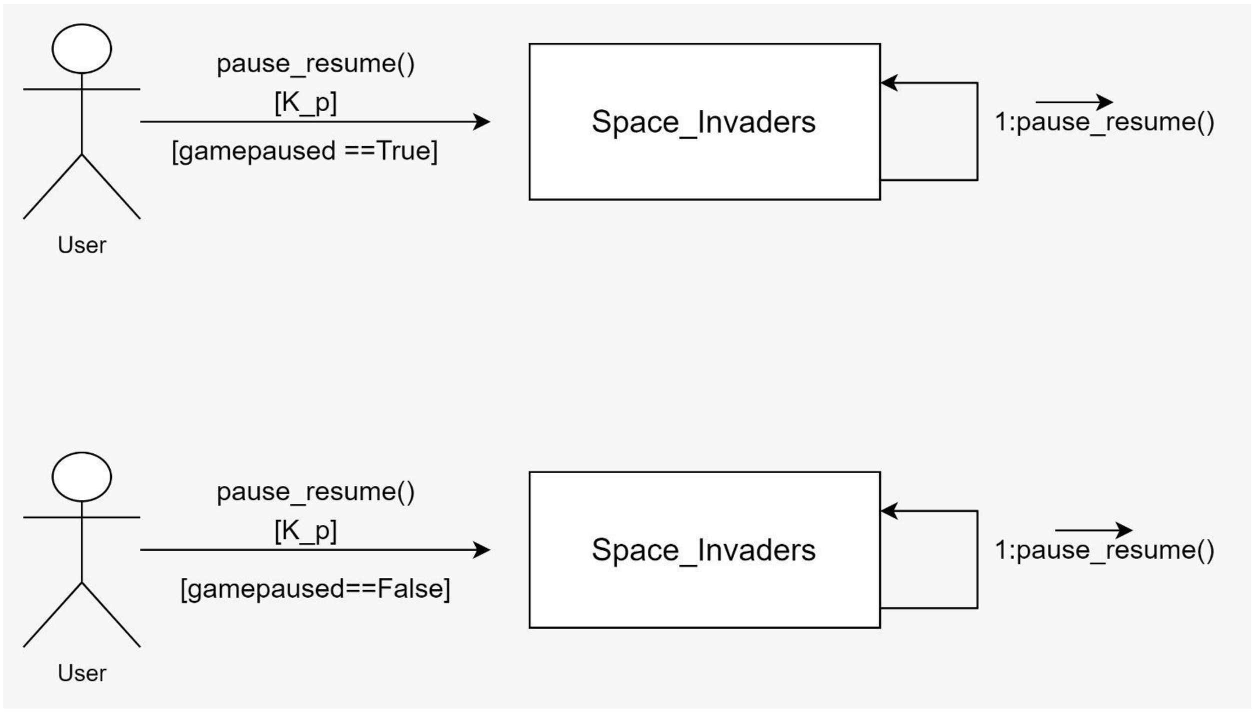
When the player wants to exit from the game, Space_Invaders class deletes all objects that game has and gets the default value of the score and number of bullets by sending a message to itself.

## 5. move_ship



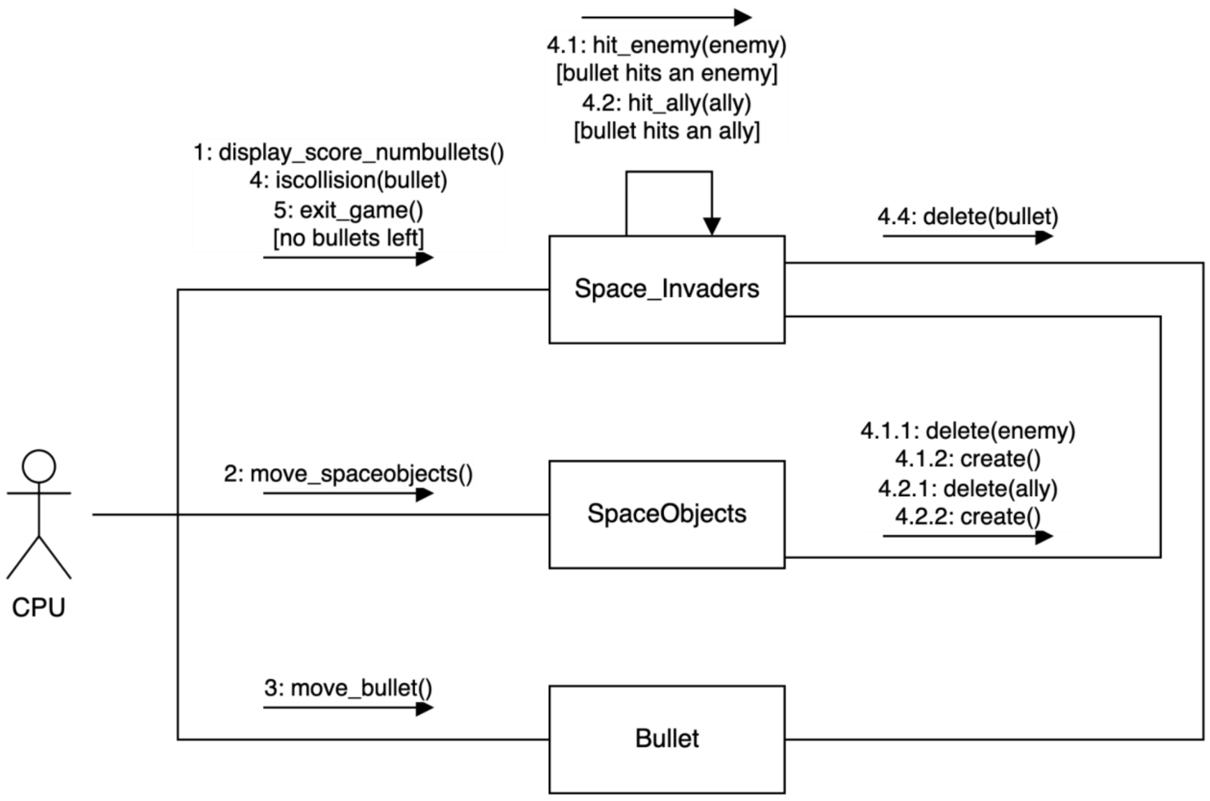
When the player initiates the movement of the spaceship, Space_Invaders class sends a message to the Spaceship class by calling the move_spaceship() function with the “change” parameter that keeps the positive or negative integer number that is determined by the left or right movement.

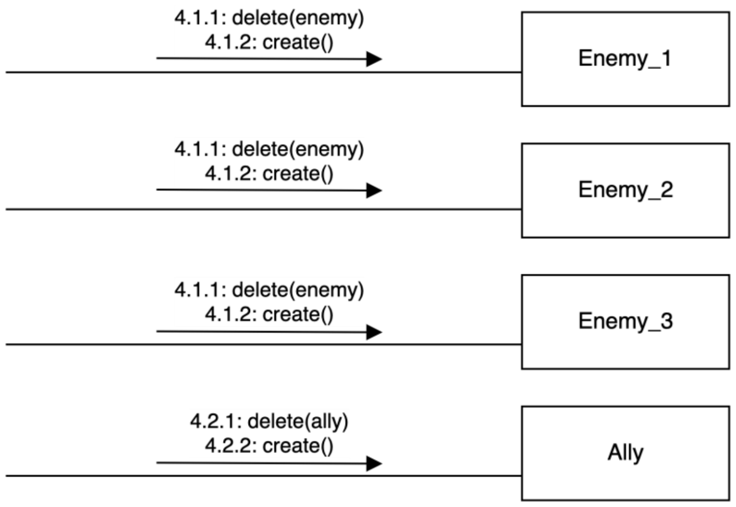
## 6. pause_resume



When the player pauses or resumes the game, movement and usage of the SpaceObjects and Spaceship will be stopped or restarted by changing the boolean variable. In the while loop movement of the objects and shoot functions are controlled by an if statement that satisfies these functions to be functional only if game is not paused.

## 7. time.tick





**Polymorphic Functions in time.tick**

time.tick is the part that is constantly ran by the CPU. After necessary objects are created at start(), computer repeatedly runs the system with the functions described below.

➢ display_score_numbullets() function always displays the instant score and number of bullets values on the screen until the game is closed.

➢ move_spaceobjects() and move_bullet() functions dynamizes the enemy, ally and bullet objects by checking their timesincelastmove variable and when it gets equal to the waitingtime variable, objects move.

➢ iscollison() function checks if are there any collisions with enemy or ally objects or not. After iscollison() function, there are 3 conditions:

- bullet hits an enemy,
- bullet hits an ally,
- bullet does not hit an enemy or an ally.

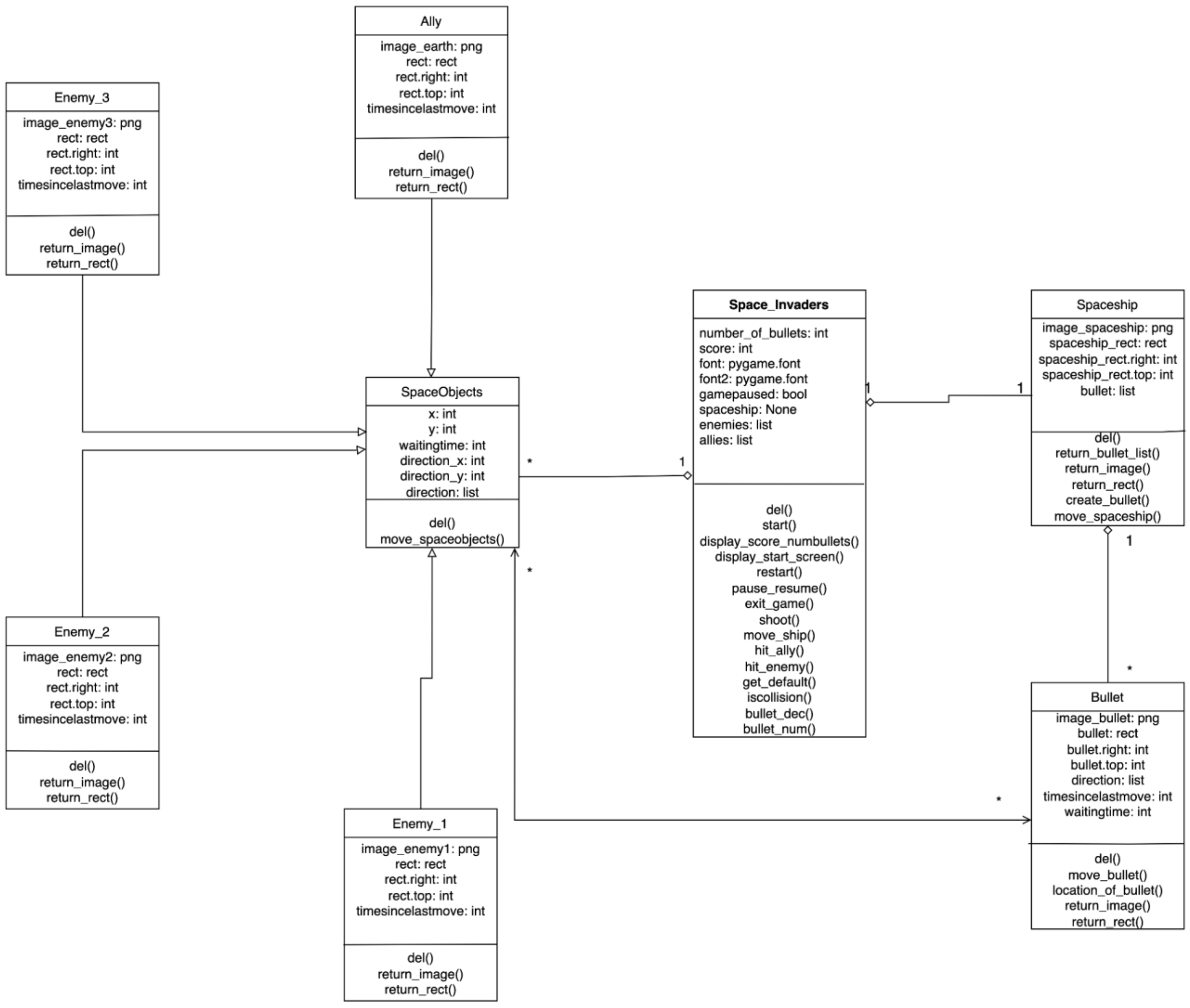
If the bullet hits an enemy, first, the bullet object is deleted with the delete() message sent to the Bullet class, and the Space_Invaders class calls itself with hit_enemy() function. Then this function increases the score by 10 and increases the bullet number by 1. Also, it deletes the enemy object by sending a delete() message to the Enemy class. After that, it creates a new enemy object to replace the deleted one by sending a create() message to the Enemy class.

If the bullet hits an ally, first, the bullet object is deleted with the delete() message sent to the Bullet class, and the Space_Invaders class calls itself with hit_ally() function. Then this function decreases the score by 10 and decreases the bullet number by 2. Also, it deletes the ally object by sending a delete() message to the Ally class. After that, it creates a new ally object to replace the deleted one by sending a create() message to the Ally class.

If the bullet does not hit an enemy or an ally before it exits the screen, the bullet object is deleted with the delete() message sent to the Bullet class when the bullet exits the screen.

➢ As the CPU's last task, if the number of bullets drops to 0, it calls the exit_game() function in the Space_Invaders class and ends the game.

## Class Diagram



## Explanation of The Classes and The Relations Among Them

### Classes:

- **Space_Invaders**

Space_Invaders class is the game class. This class receives use case messages from the user to initiate actions.
Super_Invaders class involves,

- number_of_bullets variable which stores the number of bullets that the game has
- score variable which stores the score of the player
- gamepaused boolean variable
- "enemies" and "allies" list objects that keeps enemy and ally objects.
- start() function which creates the Space Objects.
- display_score_numbullets() function which displays the score and number of bullets to the screen instantly.
- display_start_screen() function displays the start screen.
- restart() function which first stops the current game by the exit_game() function and then starts a new game by calling the start() function.
- pause_resume() function pauses or restarts the game.
- exit_game() function deletes all the game objects that the Space_Inavders class has.
- shoot() function which sends a message to Spaceship class to create a bullet object.
- move_ship() function sends a message to Spaceship class to start its movement.
- hit_enemy() function: when a bullet hits an enemy, it increases the score by 10 and increases the bullet number and deletes the enemy and bullet objects. Then creates a new random enemy object.
- hit_ally() function: when a bullet hits an ally, it decreases the score by 10 and decreases the bullet number by 2 and deletes the ally and bullet object. Then creates a new ally object.

- get_default() function: gets the default values of score and number of bullets.
- iscollision() function which determines whether a bullet misses or hits an enemy or ally and calls one of the hit_enemy() or hit_ally() functions in Space_Invaders class.
- bullet_dec() function which decreases the number of bullet by 1 when a bullet is created by the spaceship.
- bullet_num() function returns the number_of _bullets.
- __del__() function deletes an object.

- **SpaceObjects**

SpaceObject class is the parent class of sub enemy and ally classes. Enemies are the targets that the spaceship should hit to gain score. And allies are the space objects that the user shouldn't hit. When the user hits an enemy it gains score and when the user hits an ally it loses score.
SpaceObject class involves,

- Random location variables: x, y
- waitingtime variable which is a random integer for every SpaceObject to determine their waiting time between two moves.
- direction_x, direction_y variables for the direction of movements of SpaceObjects.
- move_spaceobjects() function that allows them to move randomly.
- __del__() function deletes an object.

- **Enemy_1, Enemy_2, Enemy_3, Ally**

These classes are subclasses of the SpaceObjects class. They each represent individual enemies or allies that the user can shoot with bullets.
Enemy_1, Enemy_2, Enemy_3, Ally classes involve,

- Location variables: rect.right, rect.top
- And an image file that contains the enemy or ally graphics for each enemy and ally individually.
- timesincelastmove variable which tracks the time that past since their last movement
- return_image function returns the image of the object to be used in the blit function
- return_rect function returns the rect version of image of the object to be used in the blit function
- __del__() function deletes an object.

- **Bullet**

The bullet is what the spaceship hits the enemies with.Bullets deal damage to enemies they hit.In our game bullets are very critical because if you run out of bullets you will lose the game.

Bullet class holds,

- An image file that contains the bullet graphics.
- bullet._right, bullet._top variables that will be determined with the location_of_bullet() function.
- There is a direction variable that is the direction of the movement of the bullet object.
- timesincelastmove variable that tracks the time has past since the last move.
- waitingtime variable that is an integer that representes the time that should past between two movements of the bullet object.
- location_of_bullet() function which determines the location of the bullet with respect to the movement of the spaceship.
- move_bullet() function gives bullet object a movement.
- return_image() function returns the image of bullet to be used in the blit function.
- return_rect() function returns the rect version of image of bullet to be used in the blit function.
- __del__() function deletes an object.

- **Spaceship**

  The spaceship is moved by the player with the arrow keys. Player has a mission to increase the score.

  Spaceship class involves that:

  - An image file that contains the spaceship graphics.
  - Location Variables: self._spaceship_rect.top, self._spaceship_rect.right variables
  - a list called "bullet" which stores the bullet objects that the spaceship has.
  - move_spaceship() function that can do movement from right to left and left to right.
  - create_bullet() function which creates Bullet objects and adds it to the "bullet" list when Space_Invaders class sends a message to do so.
  - return_image() function returns the image of the spaceship to be used in the blit function
  - return_rect() function returns the rect version of image of the spaceship to be used in the blit function
  - return_bullet_list() function return the "bullet" list.
  - location() function return the instant location of the spaceship in order to be used in the locetion_of_bullet() function in Bullet class.
  - __del__() function deletes an object.

## User Documentation

Just after we run the program, we first see a text telling us to press the "enter" key to start the game. We start the game by pressing enter. In the game, the user controls a spaceship by using the directional keys and tries to kill the alien enemies that come across him by pressing the spacebar key. The spacebar key allows us to fire bullets from the spaceship. For each bullet that hits the aliens, one more bullet is added to our ammo, and the number of bullets we have remains constant. Enemies are not the only space objects that come across, but also earth-shaped objects come across. When the bullet we fired from our spaceship hits the earth, two more bullets vanish from our ammunition. And when the number of bullets in our ammunition becomes zero, the game ends. The score logic in the game works as follows, for each alien we hit, our score increases by 10, and for each earth we hit, it decreases by 10.

There are some shortcut keys that we can use while playing the game. If we press the "p" key, the game stops and after we press the “p” key again, it continues to flow. To restart, we use the "r" key. When we press the "e" key, we exit the game and go to the starting screen.