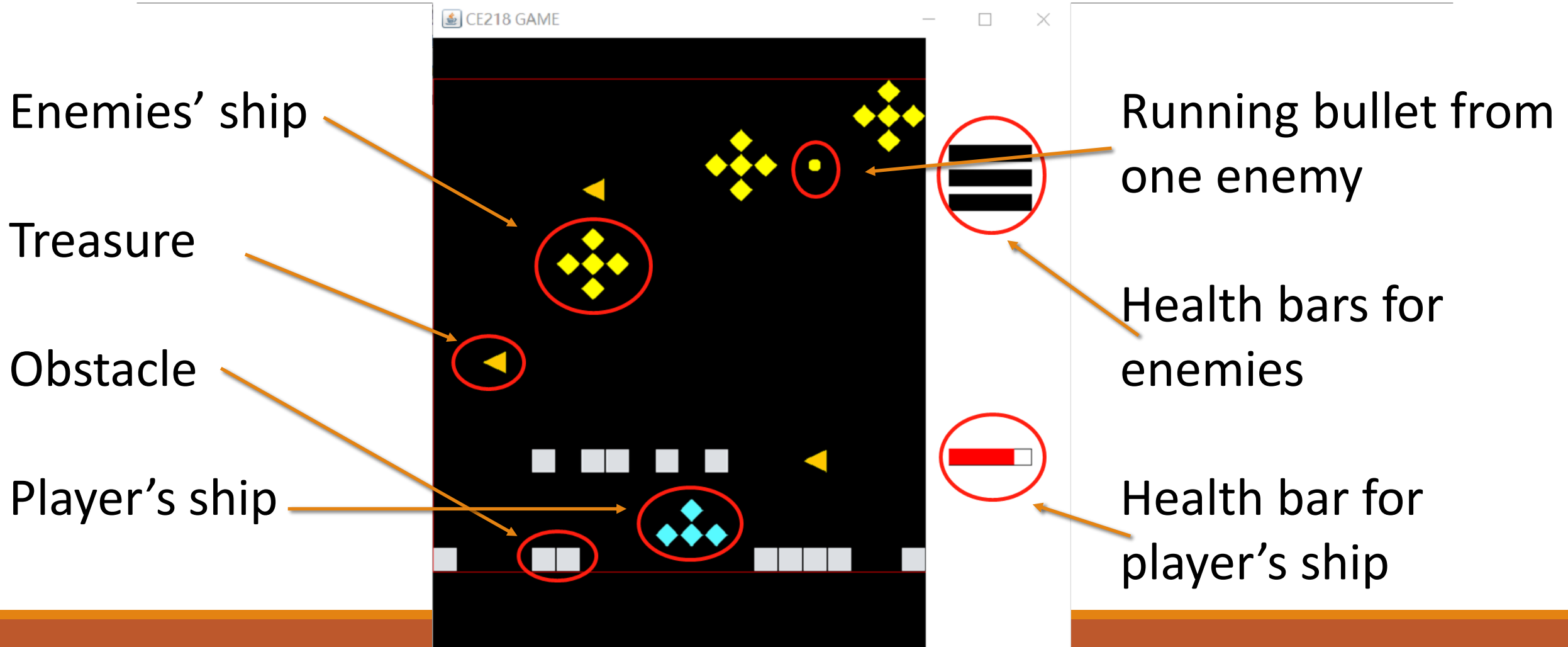


# Basic introduction for my game:

---

- 1) It is obviously only for one player.
- 2) There are several entities in the game:
  - Player's ship, robot-controlled enemies' ships,
  - Fired bullets,
  - Moving obstacles,
  - Randomly generated treasures
- 3) Rules of the game:
  - A. player' ship could move and fire in different directions as well as enemies' ships
  - B. obstacles is moving and may hurt player's ship, reducing its health
  - C. treasures could be collected by the player that then gain benefits
  - D. enemies' health reduce only when hit by player, but all enemies could hit player's ship
  - E. when player's health reduce to 0 or all enemies are eradicated, game is over
  - F. the running distance of bullets is limited, but bullets could go through obstacles

# Basic introduction for my game:



# Report of "Space" ("Sci-Fi") themed 2D video game

---

1. how to run and play my game
2. how to design and implement my game
3. significant parameters to tune for my game
4. UML of main classes
5. self-evaluation for my game

# 1. how to run and play my game

## How to run:

A. initialize the game field

B. create entities such as:

- the players ship, enemies' ships
- obstacles and treasures

C. launch the game with GUI

- (graphic user interface)

```
// initialize 2D-array  
GAME_MAP.init();
```

```
// add in player's ship  
Ship myship = new Ship(1, new Point2D(10, 18));  
GAME_MAP.addShip(myship);  
  
// add in obstacles  
List<Obstacle> obs_list = GAME_MAP.addObstacles(10, 17);  
  
// add in treasures  
List<Treasure> tre_list = GAME_MAP.addTreasure(3);  
  
// add in enemies' ship  
// number, how many enemies' ships are added in  
// 2-6 is suggested  
int number = 3;  
Ship[] enemies = new Ship[number];  
for (int i = 0; i < enemies.length; i++) {  
    Point2D pos = GAME_MAP.getFreePos(2, 0, -10);  
    enemies[i] = new Ship(2, pos);  
    GAME_MAP.addShip(enemies[i]);  
}  
  
// health info  
GAME_MAP.healthInfo(myship, enemies);
```

```
// GUI frame created  
GUIShow frame = new GUIShow(myship, enemies, obs_list, tre_list);  
// run the game  
frame.running();
```

# 1. how to run and play my game

---

## **How to play:**

A. player's ship is controlled by keyboard

- ↑ ↓ ← → of keyboard mean move ship towards up, down, left and right
- W A D of keyboard correspond to fire in up, left and right directions

B. enemies' ships are controlled by robot

C. obstacles automatically move downwards with uniform speed

D. treasures are generated randomly but will disappear after a certain amount of time

## 2. how to design and implement my game

---

### **Design:**

#### A. entities:

- Ship that has some attributes, could move and fire
- Bullet that has some attributes, could move, fired by ships
- Obstacle that has some attributes, could move
- Treasure that has some attributes, only exists for certain duration
- GAME\_MAP that collects all the game data together

## 2. how to design and implement my game

---

### **Design:**

#### B. relationships:

- Ship – Bullet, every ship could fire many bullets, so Bullet is included in Ship
- GAME\_MAP is the basic “container” for all entities as it collect the main data of the game

#### C. GUI, run the game using 2D frame

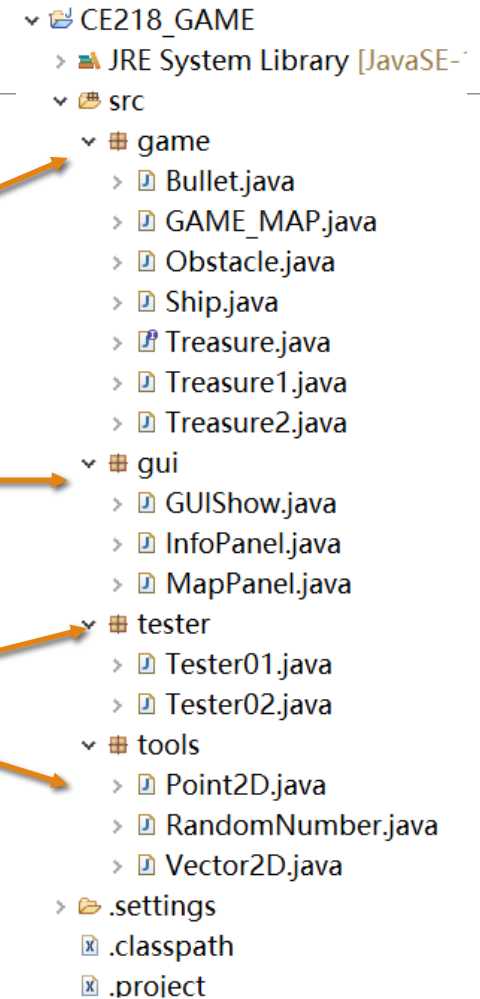
#### E. Test, testing is always necessary

## 2. how to design and implement my game

### Implement:

There are 4 packages:

- game, include the all entity classes
- gui, show the game in GUI
- tools, provide some basic functions
  - such as: random number, 2D point
- tester, used to test the game
  - both by command (Tester01) and GUI (Tester02)

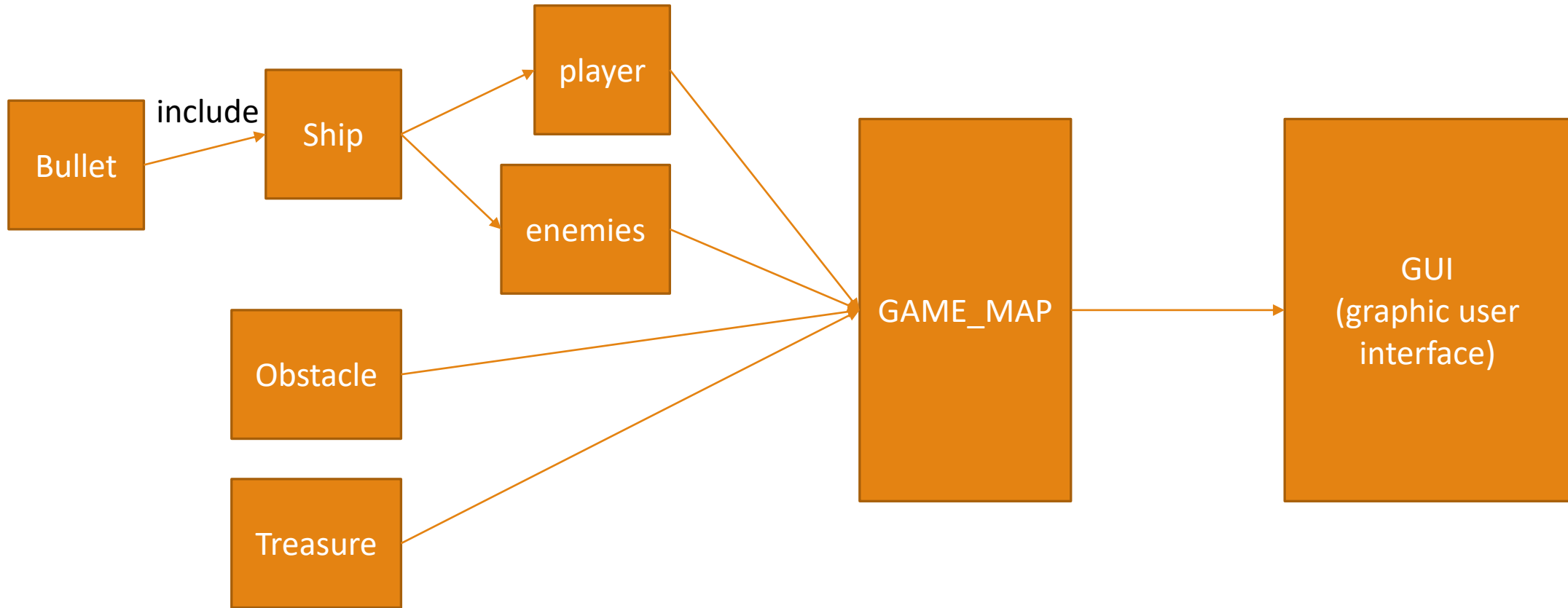




## 2. how to design and implement my game

---

The overall architecture of my game:



## 2. how to design and implement my game

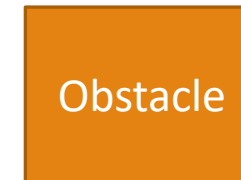
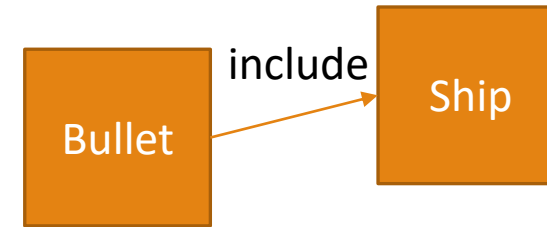
---

### Illustration in detail about implementation:

A. Ship could fire by creating new bullets, so Bullet class is included in the Ship class.

B. other entity classes relatively have an independent relationship with each other

C. all entities (Ship, Bullet, Obstacle and Treasure) have a position which is 2D info, namely, Point2D that consists of (x, y).



## 2. how to design and implement my game

GAME\_MAP

### Illustration in detail about implementation:

A. the game map is 20\*20 size that contains all the entities.

B. to implement GUI, four 2D-arrays are used to record the info about those entities.

- map for info of ships, map1 for bullets, map2 for obstacles and map3 for treasures

C. initially, set the values of all the 2D-arrays as 0. Then iterate all the 2D-arrays and set the positions occupied by certain entities as different values.

D. when drawing the frame, just iterate the 2D-arrays, if the value is 0, skip it, if not 0, draw it with different shapes and colors according to different values.

```
GAME_MAP.java
9 public class GAME_MAP {
10
11     // the size of the game map
12     public static int WIDTH = 20;
13     public static int HEIGHT = 20;
14
15     // the game map is 2D-array
16     // there are 4 maps to store the info of the game
17     // when visualizing the game, those maps should all be considered
18     // please check the GUIShow and MyPanel class
19     public static int[][] map; // used to store the positions of ships
20     public static int[][] map1; // used to store the positions of bullets
21     public static int[][] map2; // used to store the positions of obstacles
22     public static int[][] map3; // used to store the positions of treasures
23     public static int[] health; // used to store the health info of ships
24
25     public static void init() {
26         // initialize the 2D-array map
27         // 0 means it is blank
28         map = new int[HEIGHT][WIDTH];
29         map1 = new int[HEIGHT][WIDTH];
30         map2 = new int[HEIGHT][WIDTH];
31         map3 = new int[HEIGHT][WIDTH];
32         // set as 0 for free place
33         for (int i = 0; i < map.length; i++) {
34             for (int j = 0; j < map[0].length; j++) {
35                 map[i][j] = 0;
36                 map1[i][j] = 0;
37                 map2[i][j] = 0;
38                 map3[i][j] = 0;
39             }
40         }
41     }
42 }
```

## 2. how to design and implement my game

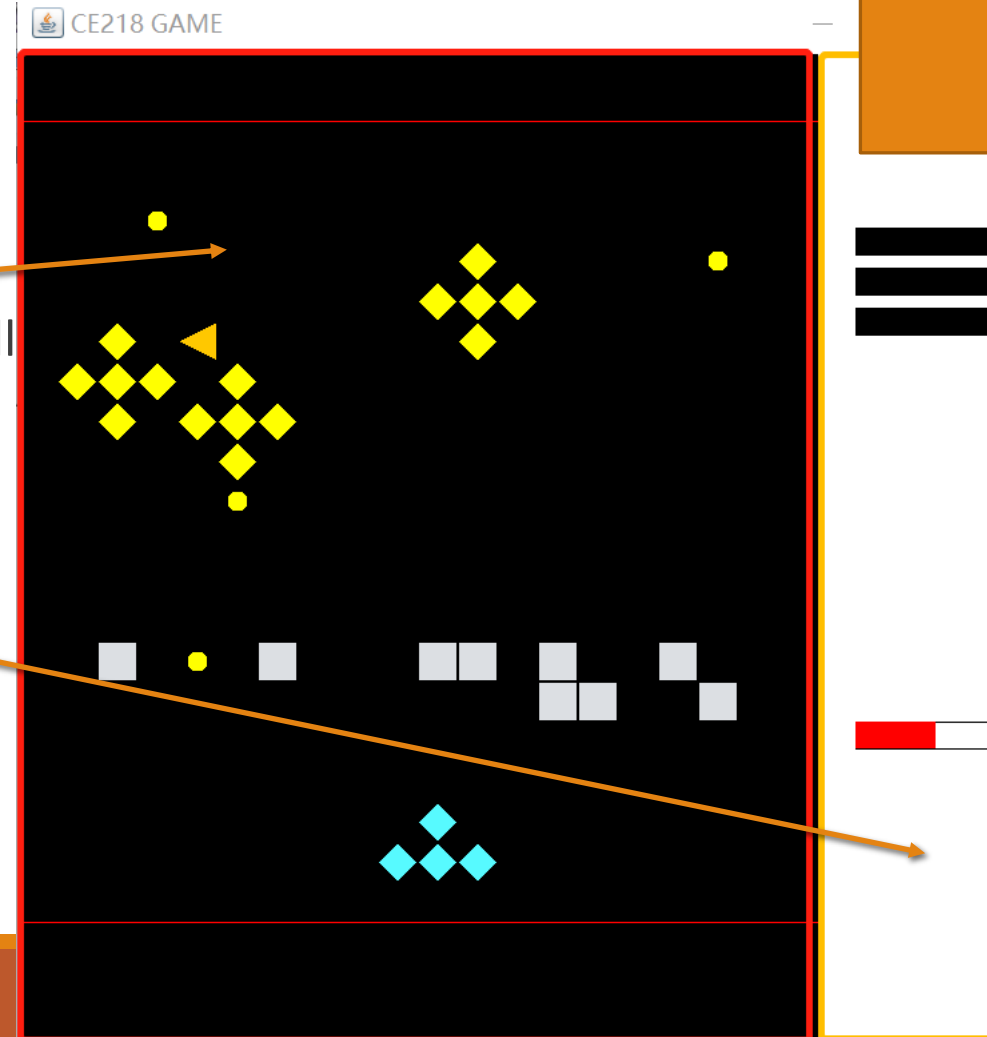
### Illustration in detail about implementation:

For GUI implementation, there are two panels on the frame to form the interface of the game.

- panel\_map (left side), used to show the field where all entities activate
- panel\_info (right side), used to show the health info about the player's ship and enemies' ships

GUIShow.java

```
19 public class GUIShow {
20
21     // size of the frame
22     public static int WIDTH = 800;
23     public static int HEIGHT = 800;
24
25     // the frame as the basic container
26     // there are two panels added on the frame
27     // one is for map, and other is for info
28     private JFrame frame;
29     private JPanel panel_map;
30     private JPanel panel_info;
```



# 3. significant parameters to tune for my game

When playing the game:

(Tester02 is a sample to run the game with some parameters that matter):

**1**, means player's ship

In the Ship class it has an attribute called 'type', 1 means player's ship, while 2 is enemy' ship.

**new Point2D(10, 18)** is the position of where the ship starts. The game is 20\*20 size, (10, 18) is the center position of the bottom.

```
Tester02.java
18
19 // test GUI frame
20 // initialize 2D-array
21 GAME_MAP.init();
22
23 // add in player's ship
24 Ship myship = new Ship(1, new Point2D(10, 18));
25 GAME_MAP.addShip(myship);
26
27 // add in obstacles
28 List<Obstacle> obs_list = GAME_MAP.addObstacles(10, 17);
29
30 // add in treasures
31 List<Treasure> tre_list = GAME_MAP.addTreasures(3);
32
33 // add in enemies' ship
34 // number, how many enemies' ships are added in
35 // 2-6 is suggested
36 int number = 3;
37 Ship[] enemies = new Ship[number];
38 for (int i = 0; i < enemies.length; i++) {
39     Point2D pos = GAME_MAP.getFreePos(2, 0, -10);
40     enemies[i] = new Ship(2, pos);
41     GAME_MAP.addShip(enemies[i]);
42 }
43 // health info
44 GAME_MAP.healthInfo(myship, enemies);
45
46 // GUI frame created
47 GUIShow frame = new GUIShow(myship, enemies, obs_list, tre_list);
48 // run the game
49 frame.running();
```

### 3. significant parameters to tune for my game

When playing the game:

When initializing the 'Obstacles', because Obstacle is generated randomly, it should be given a range where obstacles' positions should be limited.

**(10, 17)** means: obstacles could be generated within rows from 10 to 17 (the lower-half part of the game map)

```
Tester02.java
18
19 // test GUI frame
20 // initialize 2D-array
21 GAME_MAP.init();
22
23 // add in player's ship
24 Ship myship = new Ship(1, new Point2D(10, 18));
25 GAME_MAP.addShip(myship);
26
27 // add in obstacles
28 List<Obstacle> obs_list = GAME_MAP.addObstacles(10, 17);
29
30 // add in treasures
31 List<Treasure> tre_list = GAME_MAP.addTreasure(3);
32
33 // add in enemies' ship
34 // number, how many enemies' ships are added in
35 // 2-6 is suggested
36 int number = 3;
37 Ship[] enemies = new Ship[number];
38 for (int i = 0; i < enemies.length; i++) {
39     Point2D pos = GAME_MAP.getFreePos(2, 0, -10);
40     enemies[i] = new Ship(2, pos);
41     GAME_MAP.addShip(enemies[i]);
42 }
43 // health info
44 GAME_MAP.healthInfo(myship, enemies);
45
46 // GUI frame created
47 GUIShow frame = new GUIShow(myship, enemies, obs_list, tre_list);
48 // run the game
49 frame.running();
```

# 3. significant parameters to tune for my game

When playing the game:

When initializing the 'Treasures', here only **3** treasures are added in to the map.

For enemies, here is the number of how many enemy ships to be created. Here it is only **3**. (the number should not be too big when considering the capacity of computation.)

```
Tester02.java
18
19 // test GUI frame
20 // initialize 2D-array
21 GAME_MAP.init();
22
23 // add in player's ship
24 Ship myship = new Ship(1, new Point2D(10, 18));
25 GAME_MAP.addShip(myship);
26
27 // add in obstacles
28 List<Obstacle> obs_list = GAME_MAP.addObstacles(10, 17);
29
30 // add in treasures
31 List<Treasure> tre_list = GAME_MAP.addTreasure(3);
32
33 // add in enemies' ship
34 // number, how many enemies' ships are added in
35 // 2-6 is suggested
36 int number = 3;
37 Ship[] enemies = new Ship[number];
38 for (int i = 0; i < enemies.length; i++) {
39     Point2D pos = GAME_MAP.getFreePos(2, 0, -10);
40     enemies[i] = new Ship(2, pos);
41     GAME_MAP.addShip(enemies[i]);
42 }
43 // health info
44 GAME_MAP.healthInfo(myship, enemies);
45
46 // GUI frame created
47 GUIShow frame = new GUIShow(myship, enemies, obs_list, tre_list);
48 // run the game
49 frame.running();
```

# 3. significant parameters to tune for my game

## When playing the game:

When creating ship, a position should be generated randomly from the map.

The shape of the ship should be considered, because the generated position should have enough space to harbor the ships with different size.

**(2, 0, -10): 2** means the type of ship is enemy that has different shape from player's.

**0, -10** are the offset of the position that is ranged from (0, 0)-(19,19), with the offset (0, -10), then it is (0, 0) – (9, 9), suggesting the enemy ships start on the upper-half of the game map.

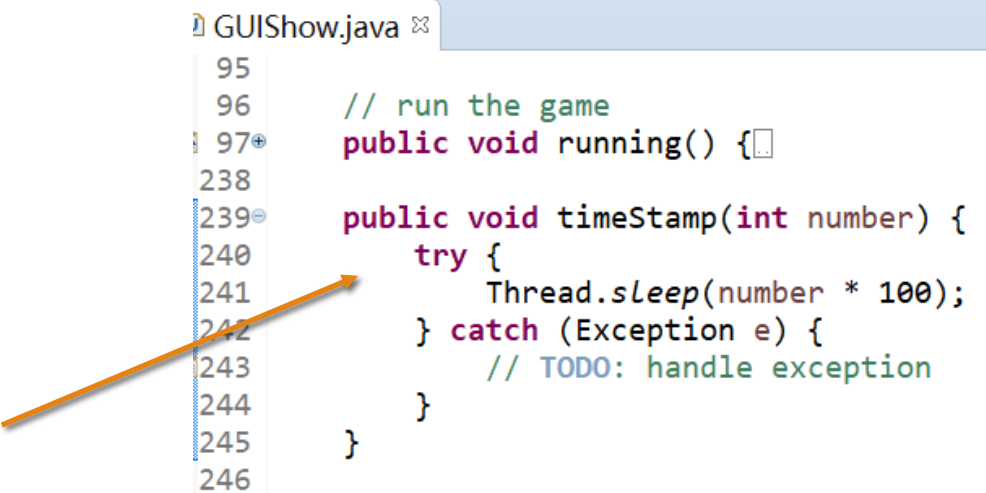
```
Tester02.java
18
19 // test GUI frame
20 // initialize 2D-array
21 GAME_MAP.init();
22
23 // add in player's ship
24 Ship myship = new Ship(1, new Point2D(10, 18));
25 GAME_MAP.addShip(myship);
26
27 // add in obstacles
28 List<Obstacle> obs_list = GAME_MAP.addObstacles(10, 17);
29
30 // add in treasures
31 List<Treasure> tre_list = GAME_MAP.addTreasures(3);
32
33 // add in enemies' ship
34 // number, how many enemies' ships are added in
35 // 2-6 is suggested
36 int number = 3;
37 Ship[] enemies = new Ship[number];
38 for (int i = 0; i < enemies.length; i++) {
39     Point2D pos = GAME_MAP.getFreePos(2, 0, -10);
40     enemies[i] = new Ship(2, pos);
41     GAME_MAP.addShip(enemies[i]);
42 }
43 // health info
44 GAME_MAP.healthInfo(myship, enemies);
45
46 // GUI frame created
47 GUIShow frame = new GUIShow(myship, enemies, obs_list, tre_list);
48 // run the game
49 frame.running();
```



# 3. significant parameters to tune for my game

## Other parameters that matter:

- Ship: the 'health' attribute with default value 10
- Bullet: the 'fatality' attribute with default value 1, reducing health by 1 if hitting ship
- GUIShow: time-stamp, which means how often all the data will be updated. In this game, the time-stamp is set as 0.2 second, suggesting it keeps updating data and running the game per 0.2 second, such as the enemy ship will make new move every 0.2 second.

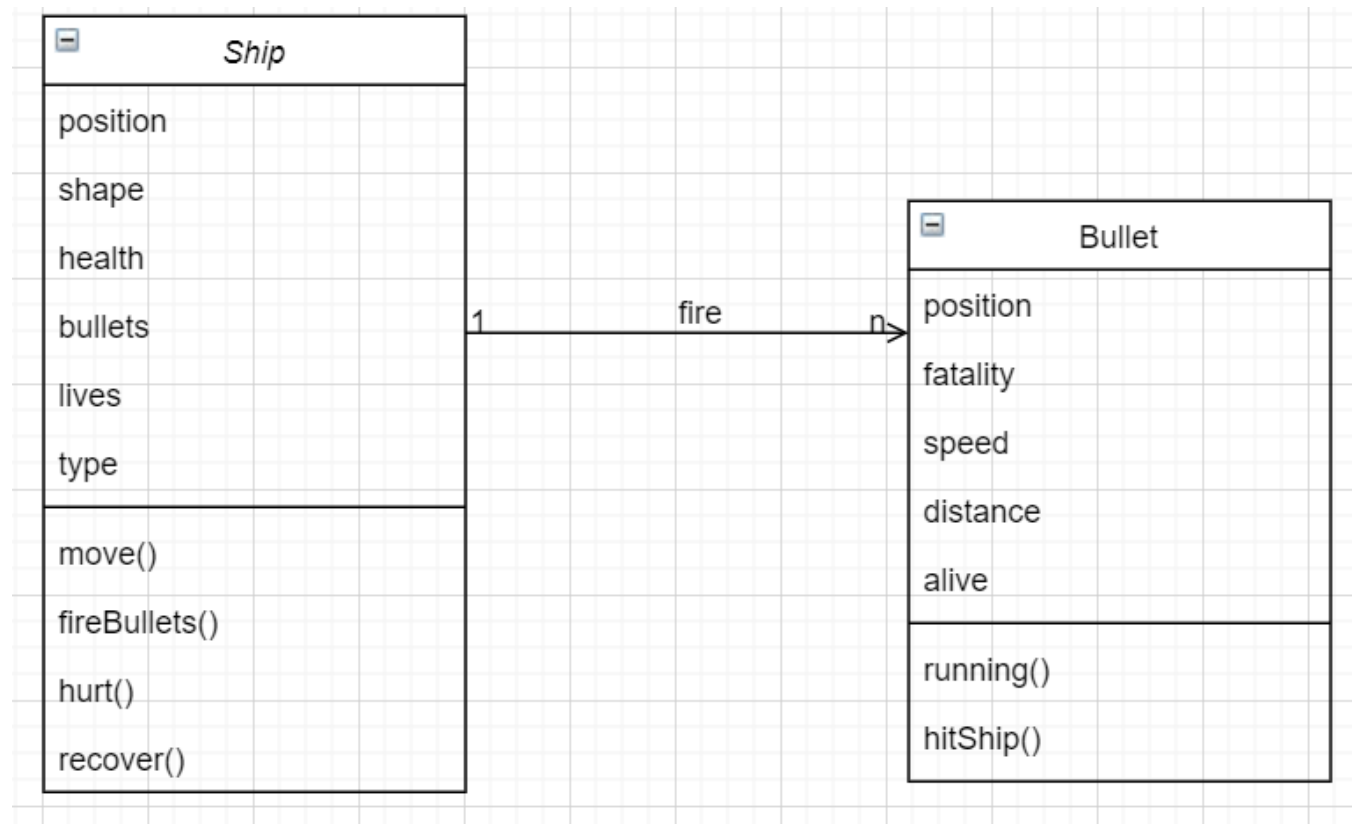


```
GUIShow.java
95
96 // run the game
97 public void running() {
238
239 public void timeStamp(int number) {
240     try {
241         Thread.sleep(number * 100);
242     } catch (Exception e) {
243         // TODO: handle exception
244     }
245 }
246
```

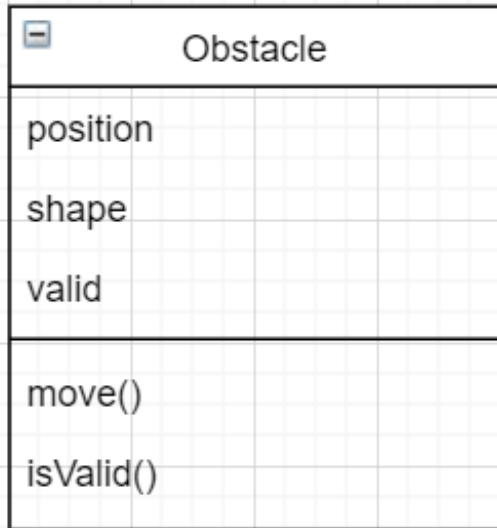
## 4. UML of main classes

---

### Ship and Bullet classes

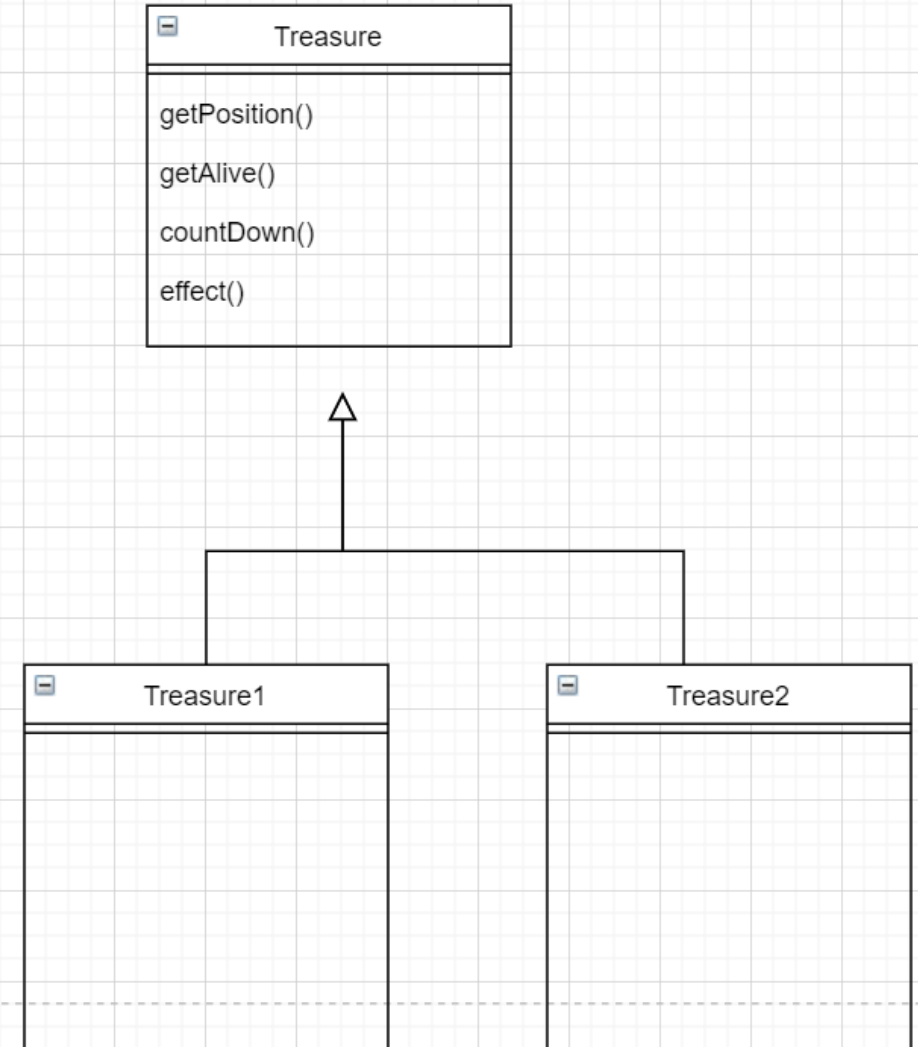


## 4. UML of main classes



Obstacle class

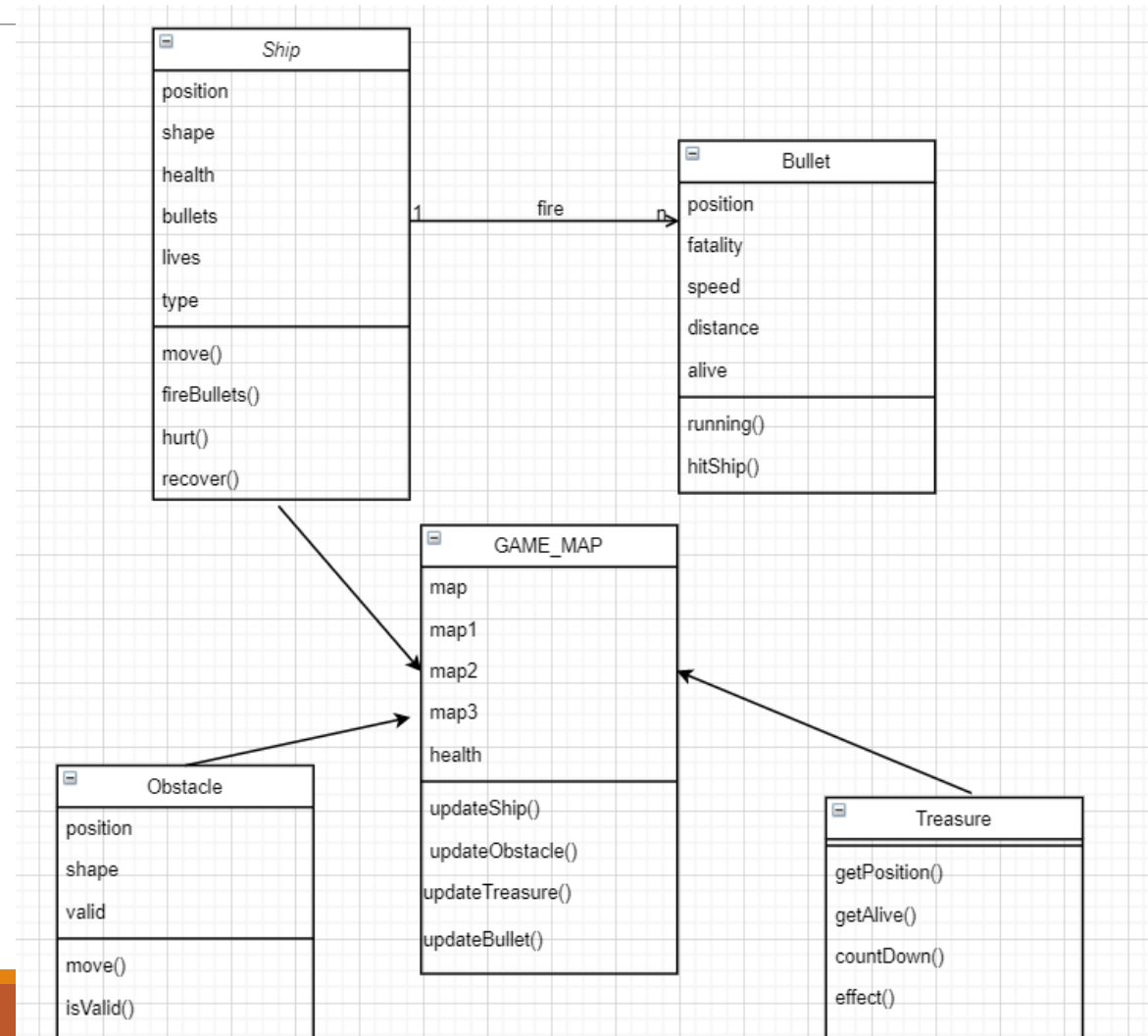
Treasure Interface  
with 2 classes to  
implement it



## 4. UML of main classes

GAME\_MAP class

And the whole architecture



# 5. self-evaluation for my game

---

- 1) Obviously, the requirements of the 'Part A' are all met
  - A. Space ship is pretty easy to move by '↑ ↓ ← →' of keyboard
  - B. Speed of Bullets is 1 grid / time-stamp (game field as 20\*20 grids)
  - C. Distance of Bullets is by default 9 grids
  - D. Enemy ships all are robot-controlled to move and fire
  - E. Game rules are illustrated before
  - F. Player's ship by default has 3 lives with 10 health points for every life

## Part A: Game Basics↵

The first part of the assignment is to get some basic, single-player game mechanics working to a high standard. Expectations for this part are:↵

- An easily manoeuvrable space ship↵
- Bullets with appropriate velocity and time/ distance to live↵
- Basic enemy ships or canons that shoot bullets↵
- Proper game rules (loss of life, scoring of points)↵
- Multiple lives and/ or multiple game levels↵

## 5. self-evaluation for my game

---

2) then, for the suggestions of 'Part B', some of them are implemented:

- A. Visual effect by using the GUI (frame)
- B. Power-up objects such as treasure, which could benefit the player's ship by recovering health and increase its fatality of bullets

3) actually, another interesting feature is added:

- A. Obstacles are moving and could hurt player's ship when hitting player,

This feature increases the interest of the game by making it more difficult to win for player.

## 5. self-evaluation for my game

---

- 4) For the design of this game project, my design is very clear and relatively flexible by following “High Cohesion & Low Coupling”, such as:
- A. all entities are very cohesive and not dependent on each other
  - B. the GUI is very independent of entities and they are connected by data not methods.
  - C. the use of interface ‘Treasure’ could also increase the flexibility and make the game much extensible if necessary
- 5) Good style of programming:
- A. divisive packages and classes and understandable naming.
  - B. comments are very detailed
  - C. classes and methods are all very cohesive

## 5. self-evaluation for my game

---

6) Some parts that could be improved:

A. obviously, the time for this project is pretty limited, if there is more time it is believed that my game could have more features and become more interesting.

B. in this game, enemy ships are randomly and stupidly move and fire, if more time is given, some AI algorithms may be tried.

C. not use threads, in this game because it is only for one player, a while-loop instead of thread is applied which is enough for this game, showing the simplicity. However, if more time is given, I would try to use threads for much smarter enemy ships that are powered by AI algorithm.



# 5. self-evaluation for my game

---

## 7) Summary:

I have learned a lot from this project, from analysis, design, coding to testing. And I believe I could do better and better for the future software projects.

---

Thank you!

