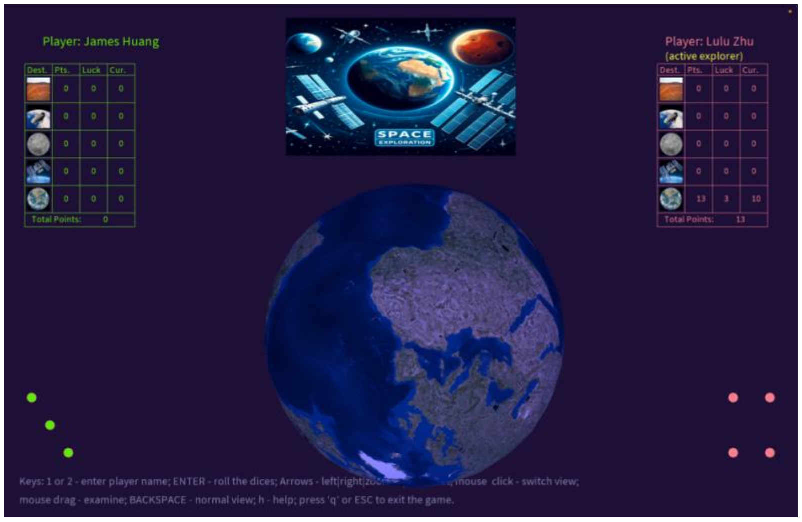
CSCI-UA. 101

INTRODUCTION TO COMPUTER SCIENCE

FALL 2023 PROJECT REPORT

Space Exploration

James Huang, Lulu Zhu



Player: James Huang

Dest.	Pts.	Luck	Cur.
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	0	0	0
Total Points:	0		

Player: Lulu Zhu
(active explorer)

Dest.	Pts.	Luck	Cur.
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	13	3	10
Total Points:	13		

Keys: 1 or 2 - enter player name; ENTER - roll the dices; Arrows - left|right|zo[illegible] mouse click - switch view;
mouse drag - examine; BACKSPACE - normal view; h - help; press 'q' or ESC to exit the game.

Instructor: Gizem Kayar Ceylan, PhD.

December'23

SECTION 1 - INTRODUCTION

Game Idea

Two adventurous players roll dices to compete and explore space destinations all the way from planet Earth to Mars in an immersive 3D deep blue space, with Hans Zimmer's *Cornfield Chase* from *InterStellar* as background music.

Background & Justification

The deep space is fascinating. Space exploration captures the imagination and interest of people around the world. Developing an immersive 3D game that allows players to explore detailed 3D rendered environments spanning from Earth, the International Space Station (ISS), the Moon, Starship crafts, to the surface of Mars would be an engaging way to bring the excitement of space travel to life and result in an educational, exciting, visually compelling game centering around humanity's journey into space.

Game Scenario:

1. Two players open the game, and see the welcome screen and an orbiting Earth, with score boards in top corners and keymap info displays in the bottom. The music starts. They press '1' and '2' to enter names, which default as "A" and "B"

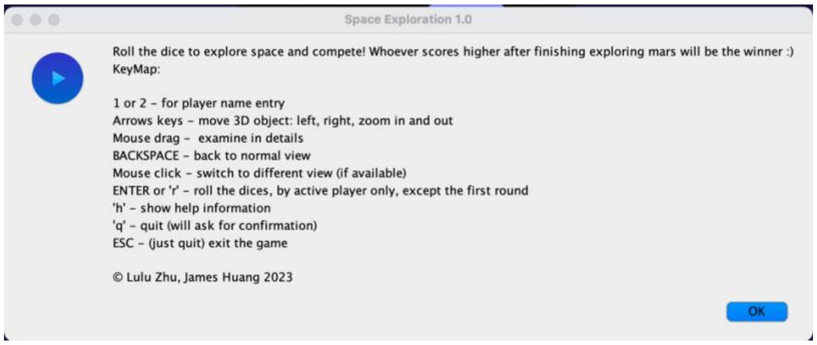
2. When ENTER is pressed, the game starts the dices start rolling. The winner earns 3 "Luck" points and advances to next destination, in the order of Earth, ISS, Moon, Starship and Mars. When exploring, the active player is rewarded with "Curiosity" points if showing curious actions by clicking on different views, moving object left|right, zooming in|out, dragging and rotating, and returning to normal view

3. The active player pressed ENTER key after finishing exploring one destination, the dice roll decides which player to be active. The games loops at step 2.

4. Whoever finishes exploring Mars and scores higher wins. After a game is over, a player can press ENTER again to start a new game. During the game, a player can press 'h' for help; or press 'q' or ESC to quit the game.

Gameplay Details

- Game Objectives:
    - Learning: to educate and inspire curious minds about the wonders of space exploration in an immersive 3D space environment
    - Fun: to entertain the players/learners with a friendly competition setting
- Game Genre:
    - a 3D adventure game in the field of Popular Science
- Game Rules:
    - 2 players; full screen: exploration order: Earth, ISS, Moon, Starship, Mars
    - Presses ENTER to roll the dice; whoever wins will get 3 “Luck” points
    - An active player can explore the active destination to earn “Curiosity” points; needs at least 5 total points on current destination to roll for next destination
    - Whoever finishes exploring Mars with a higher score wins. Pressing ‘h’ shows controls menu:
- Game Controls:

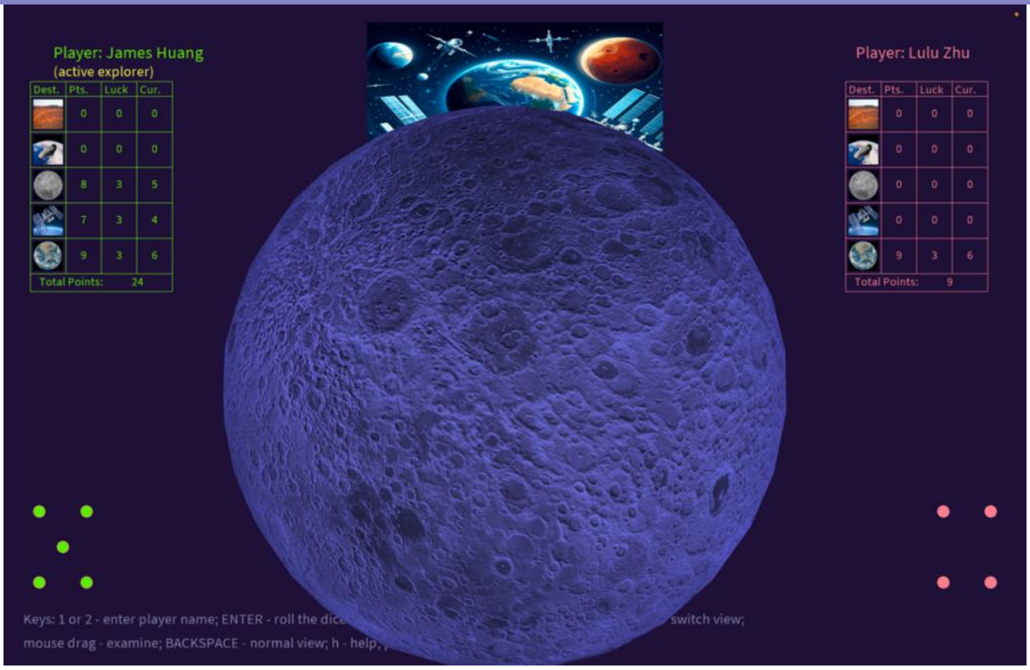


```
Roll the dice to explore space and compete! Whoever scores higher after finishing exploring mars will be the winner :)
KeyMap:

1 or 2 – for player name entry
Arrows keys – move 3D object: left, right, zoom in and out
Mouse drag –  examine in details
BACKSPACE – back to normal view
Mouse click – switch to different view (if available)
ENTER or 'r' – roll the dices, by active player only, except the first round
'h' – show help information
'q' – quit (will ask for confirmation)
ESC – (just quit) exit the game

© Lulu Zhu, James Huang 2023
```

Figure 1. Game controls.

- Game Mechanics and features:
    - 3D: an immersive 3D deep blue space
    - Rolling “lucky” dices: players compete by rolling for “Luck” points
    - “Curiosity” points: players get rewarded by showing “curiosity” action
    - Surrounding music: Hans Zimmer’s *Cornfield Chase*

SECTION 2 - IMPLEMENTATION


Player: James Huang
(active explorer)

Dest.	Pts.	Luck	Cur.
	0	0	0
	0	0	0
	8	3	5
	7	3	4
	9	3	6
Total Points:	24		

Player: Lulu Zhu

Dest.	Pts.	Luck	Cur.
	0	0	0
	0	0	0
	0	0	0
	0	0	0
	9	3	6
Total Points:	9		

Keys: 1 or 2 - enter player name; ENTER - roll the dice switch view;
mouse drag - examine; BACKSPACE - normal view; h - help,

Figure 2. Game screen of Space Exploration

Class Diagram (UML)

Shown in Figure 3 is a class diagram of the game files. We have learnt to code up using plantUML, a popular UML modeling tool with online references at https://plantuml.com/en/class-diagram. In the bottom right corner, there is a Legends class diagram shows public, private, protected, default access level data and methods' legends.

The main sketch is an unnamed class that inherits from PApplet. It has two main inherited methods: setup() and draw(). Code in setup() runs once unless explicitly called again, so I used this section to set up the full screen mode, initializes the player instance, reading and loading images files, read and play the mp3 file. Code in draw() refreshes 60 times per second, so I used this section to paint the on screen elements such as the logo, the rolling dices, the score report sections, the on-screen help text, the welcome screen, etc. The main sketch class declares and initializes 2 instances of class `Player`. So, the relationship here is 1:2 composition. The main sketch class also declares and initializes a list of `Destination` class instances, in a 1:many composition relationship. The `Player` class declares and initializes a `Dice` instance, which is 1:1 composition. The `Player` class also has declaration and initialization of a `ScoreRecord` class object, so again 1:1 composition. The `ScoreRecord` class declares and initializes a `Hashtable<String, DestinationRecord>` of `DestinationRecord` class instances, which is 1:many composition.

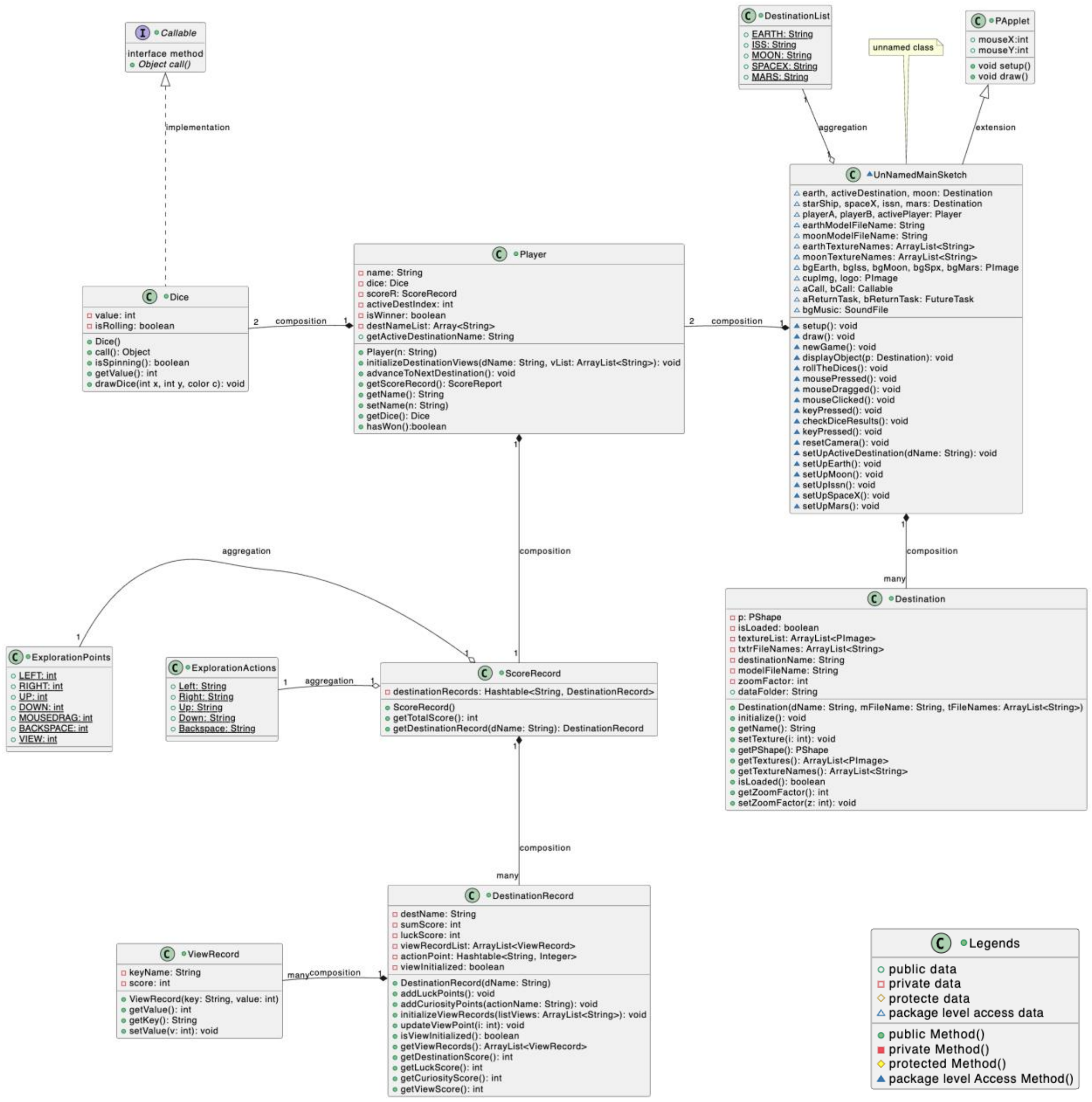


Figure 3. Class Diagram of Space Exploration. (Legends in the right bottom corner.)

`DestinationRecord` class declares and initializes a list of `ViewRecord` class instances in a 1:many composition relationship.The `ScoreRecord` class also uses static classes `ExplorationActions` and `ExplorationPoints` for lookup purposes, such relationship is 1:1 aggregation relationship as `ScoreRecord` and this classes are independent of each other.

A notable class is `Dice`, the class that is responsible for generating a number between 1-6 upon rolling. It implements Java's `Callable` Interface, which we will discuss below.

List of Techniques:

- **3D rendering and camera angle**

In main sketch class, we first initialize the fullscreen in 3D mode in `setup():`

```
fullScreen(P3D);
```

then use processing's `shape()` method to a `PShape()` 3D object in this method:

```
void displayObject(Destination p)
```

The main 3D camera angle, rotating, and zoom logic are in `displayObject()` and `resetCamera().`

- **Data Structures**

`ArrayList` and `Hashtable` are used extensively. `ArrayList` is suitable for a list whose size changes dynamically. For example, when storing a 3D model's textures, since each model can have different number of textures, `ArrayList` is ideal for this case:

```
ArrayList<String> earthTextureNames = null;

…

    earthTextureNames = new ArrayList<String>();

    earthTextureNames.add("Ocean_Mask.png");
```

`Hashtable` is suitable for (key, value) pair storage and fast retrieval by key. For example, we used a `Hashtable` to store `DestinationRecord` with each `Destination`'s name as key, this way the program can retrieve the record by name, with a constant O(1) time.

```
private Hashtable<String, DestinationRecord> destinationRecords;
```

```
  public ScoreRecord()
  {
    destinationRecords=  new Hashtable<String, DestinationRecord> ();
    destinationRecords.put(DestinationList.EARTH, new DestinationRecord(DestinationList.EARTH));
```

- **Class Decomposition**

We have managed to use OOP to decompose the solution into different classes: Player, Dice, ScoreRecord, Destination, DestinationRecord, etc. This way the logic is quite clear, which greatly simplifies the design. Plus, we get to enjoy the benefits of OOP's encapsulation, inheritance, and polymorphism.

- **Non-Blocking Client UI: Threading, Callable interface, FutureTask/ReturnTask**

When implementing the rolling dice, we have designed it this way: each dice rolls, sleeps for 0.1 second, and then rolls again for a total of 30 rounds (3 seconds). But we ran into a problem: the rolling-sleep-rolling logic would freeze up the main UI. After some research, we have realized that the logic uses the same thread as the GUI rendering thread. Therefore, we have learnt to use threading to enable parallel tasks. This way these background threads freely roll the Dices while the GUI stays responsive and displays these dices' changing values dynamically. So, we have made the Dice implements the `Callable` Interface:

```
public class Dice implements Callable<Object> {

  public Object call() throws Exception { … }
```

Then, kick off the dice rolling on a different thread:

```
  aCall = playerA.getDice();

  aReturnTask = new FutureTask(aCall);

  Thread threadA = new Thread(aReturnTask);

  threadA.start();
```

Further study led us to use `FutureTask` and `ReturnResult` to retrieve the result:

```
If (aReturnTask.isDone())

        Integer aResult = (Integer)aReturnTask.get();
```

After these code pieces are in place, the rolling dices and GUI work nicely together.

- **Exception Handling**

On I/O and thread logic, `try{}` and `catch{}`are used to catch/handle the exceptions.

SECTION 3 - CONCLUSION

Implementation Related Difficulties:

We have encountered several difficulties during the implementation. The first is that processing only supports .obj 3D model files, but such formats are rarely available in NASA’s 3D download page or other space related 3D model sites. Eventually we have found a walkaround to convert other formats like .3ds, .stl, etc to .obj format using a 3D modelling tool called Blender. The second is that free 3D models are typically of low resolution, the high-resolution ones are beyond freshman students’ budget – time may resolve this. The third one was the GUI freezing issue which we managed to find a solution (already covered in last section).

Overall, we have found this project to be quite a positive and fun learning experience because we able to apply the programming skills and design principles from Professor Gizem Kayar Ceylan’s excellent class in the related problem-solving space.

Potential Further Improvements

- Increase a variety of destinations the user could potentially visit, for example: Mercury, Venus, Galaxies, even Blackhole, etc. This will entail a larger dataset of models to be downloaded from the NASA 3D resource site.
- Add descriptions with educational purposes that explains each destination with statistics such as age, mass, diameter, such information can be acquired through NASA website, or Wikipedia.
- Add sound effects for different interaction between the player and the game might improve the engagement and playfulness of the game.
- Increase the playfulness and competitiveness of the game by turning into a similar game to Monopoly. This way, more users and destinations can be incorporated. Different roles and special occasions/encountering that could combine other small games into Space Explorer that fits the purpose of the game, such as asking questions, bingo, etc. These should greatly increase the scale, effectiveness, and entertainment of the game.

REFERENCES

NASA 3D models: https://nasa3d.arc.nasa.gov/

Another 3D model site: https://sketchfab.com/3d-models

Callable and Future Tasks: https://www.javatpoint.com/callable-and-future-in-java

3D Camera angle: https://stackoverflow.com/questions/29131367/processing-changing-orientation-of-the-camera-and-fixed-text-position

Sound: https://processing.org/reference/libraries/sound/SoundFile_play_.html

3D rendering: https://processing.org/tutorials/p3d

mp3 download site:
http://tools.liumingye.cn/music_old/?page=audioPage&type=YQB&name=cornfieldchase