National Near Earth Object Preparedness Strategy and Action Plan or NNEOPSAAP

Programming Fundamentals 1

Developer Guide

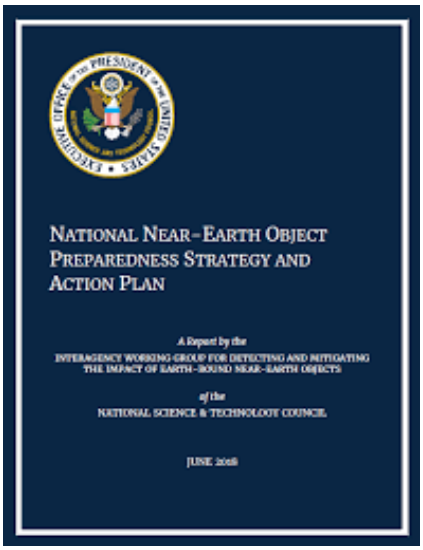
This project was approved by Furia Carlo Alberto
prof. of Programming Fundamentals 1

November 12th - December 11 2020
Universita della Svizzera italiana
Faculty of Informatics
Switzerland

Contents

1 Introduction

The idea behind the new game was to create a "space invaders"-like game that would allow a player, controlling a space ship, to shoot and eliminate approaching asteroids therefore proceed to an advance game-state which would be rewarded by the score increase. There was an intent to create an engaging and competitive environment around this game with the implementation of the leader-board which would rank players based on their score. Also, we kept internationally in mind as our leader board is hosted on an external server and therefore allows players around the world to interact and compete.



The name of the game is derived from an American defence program initiated in 2018 which would evaluate different ways to protect the Earth from asteroids and space junk in case a collision was predicted. The program was called: "National Near Earth Object Preparedness Strategy and Action Plan".

Initially the proposal for the game was to make a First-person view game which would allow a player to move around the world exploring a predefined map. Although achievable, this project would have resulted in a simple proof of concept that would be of less enjoyment to the end player. We opted to go for a more dynamic game that would result in more interesting game-play experience.

2 Project Composition

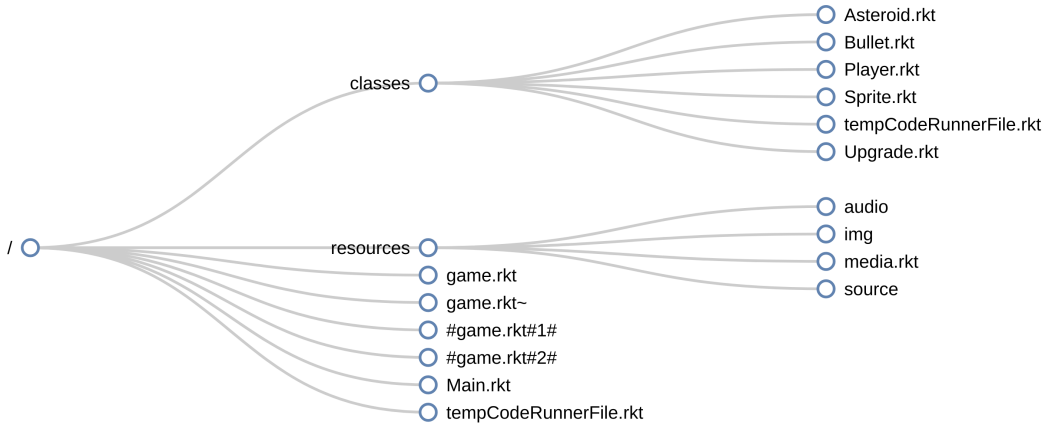
In order to make this game work we had to implement the external libraries in DrRacket:

```
(require racket/base)
(require racket/class)
(require 2htdp/universe)
(require 2htdp/image)
(require net/sendurl)
(require racket/string)
```

Considering this is a 2D game, we implemented the **Big-Bang** function that allows the player to monitor and modify the game state with all its elements. We also used this function to run other functions: draw, *handle-mouse*, *tick* and *game-end*. All the interactive graphical elements in the main menu like *buttons*, *sliders*,*textboxes* are part of the package **racket/GUI**.

draw	**handle-mouse**	**tick**
This function draws the game screen by drawing all the %sprites (player, asteroids, bullets, upgrades, CANVAS) and overlays the GUI elements (life,score) on top of it.	With the help of this function, we run the functions *movePlayer* and *shoot*. The *movePlayer* function moves the player to new coordinates and the *shoot* function creates the newly-shot bullets of the player and appends them to AppState-list-bullets.	This functions runs the following functions: *spawnElements* which spawns new %sprite objects (asteroids, upgrades) apply-speed which applies the speed of every %sprite object to it , and the function *remNotVisible* that removes any %sprite object that is not visible on screen anymore.

Structure



2.1 Data Types

All the interactive objects are part of the GUI

2.1.1 frame%

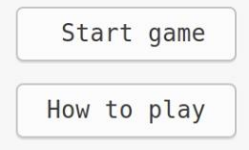
frame%	
Fields:	
main-frame	the top-level container window is opened, which displays the frame's label
sub-frame	we get a window where the player can put his name and the number of lives

2.1.2 canvas%

canvas%	
Fields:	
canvas	general-purpose window for drawing and handling events.

2.1.3 button%

button%	
Fields:	
parent, label, vert-margin, enabled, horiz-margin, callback	Whenever a button is clicked by the user, the button's callback procedure is invoked.

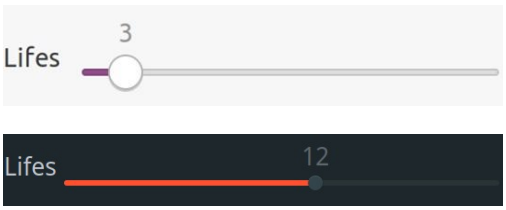


2.1.4 text-field%

text-field%	
Fields:	
name-input	a name value is given in input by the player and it's set as well as enabled the use of the Life-input slider

2.1.5 slider%

slider%	
Fields:	
label, parent, min-value, max-value, enabled, init-value,	A Panel item with a handle that the user can drag to change the control's value. The Life-input function gets the life value inputted by the player and sets it. The style of the bar changes based on the system theme.



External Files

We structured the game upon classes, allowing us to easily implement complex but easily understandable and modifiable code.
Every element on screen in the game is an Object of some %class .
We made good use of the inheritance capabilities of Racket's Class system.

2.1.6 sprite%

sprite%	
Fields:	
position : Posn	The position of the sprite
speed : Posn	X & Y speed components of the sprite
image : Image	Image of the Sprite
Methods:	
+ apply-speed : → (*void*)	increments the sprite%'s position based on the speed

The sprite% class contains all fields and methods shared by all on-screen elements. This way, functions that handle on-screen elements treat every element just as a 'sprite%' element.
The speed is multiplied by 'media.rkt's screenScale. This way the speed will always be scaled by the size of the monitor.
The class rotates the image based on the x and y speed, to make the sprite always faces the direction it's going (e.g diagonal bullets).

2.1.7 asteroid%

asteroid%	inherits: sprite%
Fields:	
Methods:	
+ hit : → (*void*)	Called when the asteroid is hit. Plays the 'hit' SFX

2.1.8 player%

player%	inherits: sprite%
Fields:	
name : String	name of the player
score : Number	score of the player
starting-lives : Number	the full player's lives
lives : Number	current player's lives
upgrades : (make-upgrades 0 0 1)	Player upgrades
shooting-pattern : List(make-bulletconf)	Contains the configurations for the bullets that get shot on (shoot)
Methods:	
+ increase-score : Number → (*void*)	Increases the score of the player by a certain amount
+ shoot : → *List*(*bullet*%)	Called when the player shoots. Plays the shooting sfx and returns the list of the bullets just shot.
+ decrease-lives : → (*void*)	Decreases the player's lives by 1
+ increase-lives : → (*void*)	Increases the player's lives by 1
+ hit : → (*void*)	Called when the player takes damage
+ heal : → (*void*)	Called when the player heals
+ add-upgrade : upgrade% → (*void*)	Extracts the name of the upgrade% given in input and adds it to the player
- addSpeedMul-upgrade : → (*void*)	Called when the player collects a 'speed multiplier' upgrade
- addFront-upgrade : → (*void*)	Called when the player collects a 'front' upgrade. Generates the new list of bulletconf 's for the front bullets
- addDiagonal-upgrade : → (*void*)	Called when the player collects a 'diagonal' upgrade. Generates the new list of bulletconf 's for the diagonal bullets

2.1.9 bullet%

bullet%	inherits: sprite%
Fields:	
Methods:	

2.1.10 upgrade%

upgrade%	inherits: sprite%
Fields:	
super[image] : Image	Sets the Image based on the upgrade% name given in input
name : String	Name of the upgrade
Methods:	
+ random-upgrade : → (*void*)	Returns a random upgrade name based on probability

3 Organization

Team Composition

Considering that we had the freedom to select project partners at our own will, we decided to undertake this project as an already established group of Friends which allowed for a better communication and work environment. The developers on this team were:

Ntwali Muhizi

Mak Fazlic

Albert Cerfeda

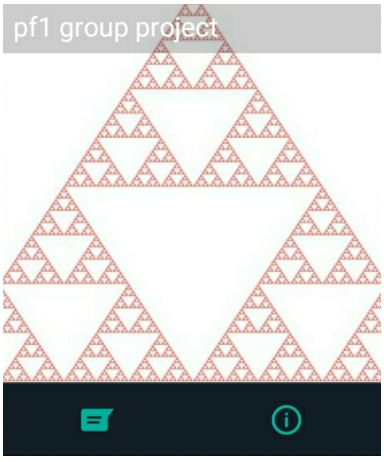
Daniela Gjorgjieva

Internal Deadlines

In order to make sure that the project is finalised with satisfactory quality, proper division of labour and assessment of time was essential. Furthermore, considering that due to the current situation most of the team meetings were moved online and proper time management was an important factor in our success. Below is presented a timetable that helped us stay on track, stay focused and stay synchronised.

Team Deadlines	Nov 9	Nov 16	Nov 23	Nov 30	Dec 7
Monday			Backbone of Game		Game-LB-UI Final
Tuesday				Leaderboard	
Wednesday			Finalising game		Documentation
Thursday	Proposal				
Friday		Division of labour	User Interface		Presentation

Communication



Whats-App/SVN/Overleaf

As a means of communication we decided to opt for Whats-App as it is the most accessible platform for all of our team members. It allows free and international communication which was circumstantially beneficial for our remote development.

The name of our group was PF1 Group Project and it contained all of the important information regarding our project. SVN was used to transfer the code remotely and securely between the team members with the help of the server provided by the faculty of Informatics.

In order to create the Documentation we decided to use overleaf as it allows excellent communication team work tools.

4 Conclusion

Overall we had a lot of fun working on this project, and with a lot of effort and work we managed to make an interesting game, each and every one of us did their part on time. With this project, we learned the fundamentals of programming and we built a good base so that we can upgrade and improve our knowledge in the future.