Master CORO M1
Master Commande et Robotique
Master in Control and Robotics

EMARO+ M1
European Master on Advanced Robotics

Project Report
24/06/2020

Developing Gorilla Game using C++ and Python

Dinh Vinh Thanh NGUYEN
Shuo YANG

Supervisor(s)

Olivier Kermorgant, Professor

Abstract

The goal of the project is to design a 1 vs 1 game with implement of a simple AI player. For our project, the classic Gorilla Game is chosen to be developed, in which each of the 2 players would try to win the game by controling a gorilla on turn-by-turn basis to throw and hit a banana to the other. For game design, the game mechanics is implemented using C++, while the game GUI is done using Python. Communication between C++ and Python to transfer messages is provided. We successfully created the Gorilaa game running smoothy in C++ and displaying well in Python/Pygame.

Table of contents

I. Introduction

1. Background

The Gorilla game project is about to develop a 1 v 1 gorilla game through communication between C++ and Python, and implement AI players in the end. In this project, C++ is responsible for all the machanics and Python is for graphic display. As far as what we can tell, most games are developed with java, C++ and other script languages. There are also some small games are developed with python language. Classic Gorilla game can be developed with only Python language, this project will focus on developing a gorilla game with pyhton and C++. Our motivation to develop this project was to practice and improve programming ability and have a peek on communication between the two interpreters.

2. Project's goals

The goal of the project is to create a game mechanics with C++ and develop a user interface which provides player input and return game feedback based the input information. The feedback information would generate state information which is sent GUI. The GUI would display different screens accroding to inital information or state information received from C++. The project also require a simple AI to realize the 1 v 1 Gorilla game.

II. Game description

1. Game’s Algorithm

The Gorilla consists of ta randomly generated enviroment where several buildings with different heights, two gorillas standing on two of these buildings. Contibuting to the enviroment are the unpredictable wind and a fixed gravity.

Given:

Initial values: Positions of two gorillas
Heights of buildings
Gravity
Radius of explosion

Changing value: Wind's direction and value

Input: Velocity of the banana shot
Angle of the banana shot

Output: Position of the landing of explosion
Boolean value of hit success
Winner of the game.

With these information, our task is to design an algorithm that allow two players change turn, throw banana shot at the enomy with freely selecting velocity and angle. It also should be able to send and update the state after every turn. Finally, it should be able to declare the winner when one is hit by the other or by itself.

A general algorithm of the game is specified as in the UML diagram below. For the Gorilla Game to work, one should expect the program does have these features following:

- The game should be able to initialize giving appropriate initial values.
- The game should be able to allow user to input and/or compute input by AI player.
- The game should be able to update states to players and send to display as well.
- The game should have a specfici result after satisfying conditions or certain playing time.

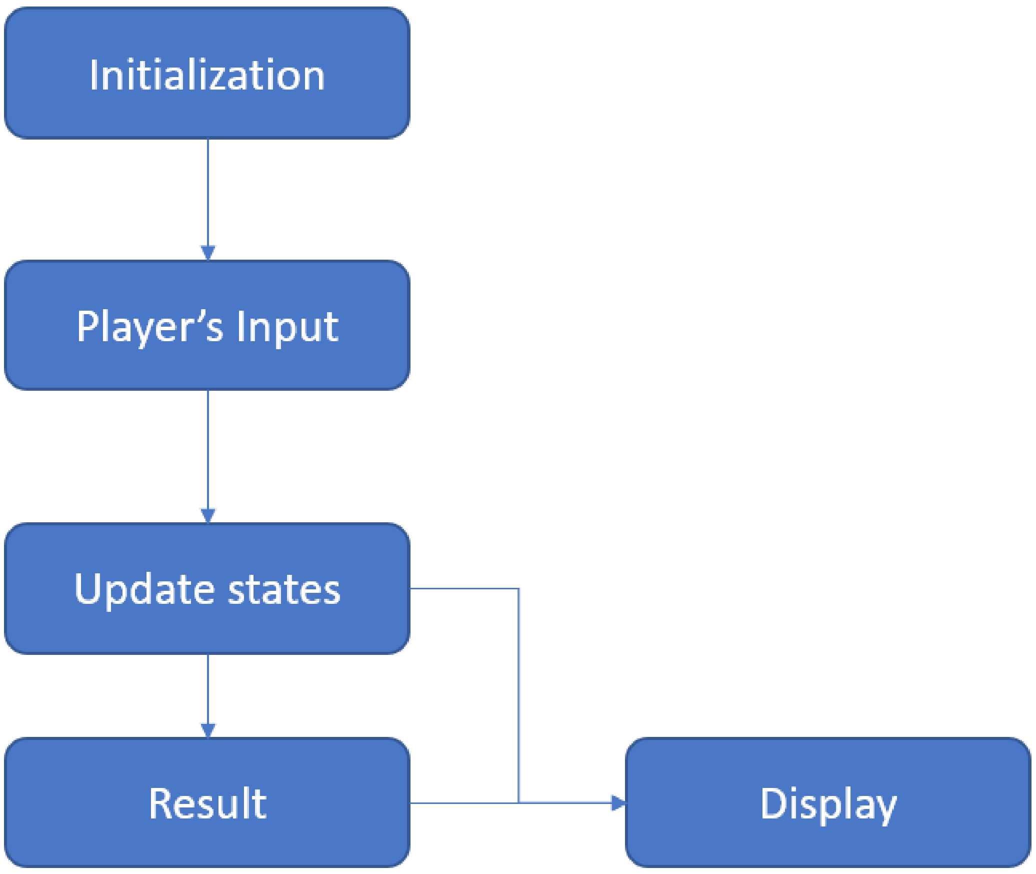


Figure 1: General diagram of Gorilla game

2. Graphical display and user interface

(a) The purpose of the graphical display component is to set up the game mode and update information after every time reading from C++. All the displaying part is based on Pygame, which is a set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming languages. The models should be displayed in this Gorilla project are two gorillas, sun, landscape, banana and explosion. Among them, the banana and explosions need to be updated at each time.

(b) The displaying part is mainly depended on machanism part(C++). what will be displayed is based on what is received by Python. Thus the basic programming structure of this part is to first read the data then decide what component or what screen need to be displayed. The reading process is to load data from a temporary file to which C++ sends the data. The first read is to get initial information, which is different from the other reads, those aiming to update states. Therefore after the first time reading, all the other reading are blocked in a while loop untill one player wins the game or exit button is pressed.

The game components and some user-defined functions are referred to another open source Gorilla game by *Al Sweigart.*

3. Communication between C++ and Python

The comnunication between the two interpreters is through a FIFO pipe, by which the first data transmitted to the pipe will be received first. However, the communication is not directly connected. The data from C++ will be transmitted to a temporary file, then Pyhton will read the data in the file. For the Python component, it reads and loads the data through a function yaml.load(). The fonction load data in yaml form and save them as dictionary in Python. That means the data sent by C++ has to satisfy yaml form, otherwise it wouldn't be loaded by yaml. If the data was not in yaml form, python will return an error complaining that it doesn't know the format. Before sending data, the temporary file should be created, otherwise the data will be sent to nowhere and python will miss some data too while reading. That is why before sending any data, the only job for C++ is to run the python code and check the state of the temporary file. The C++ code will wait in a while loop until the file is created.

III. Game design

1. Game mechanics

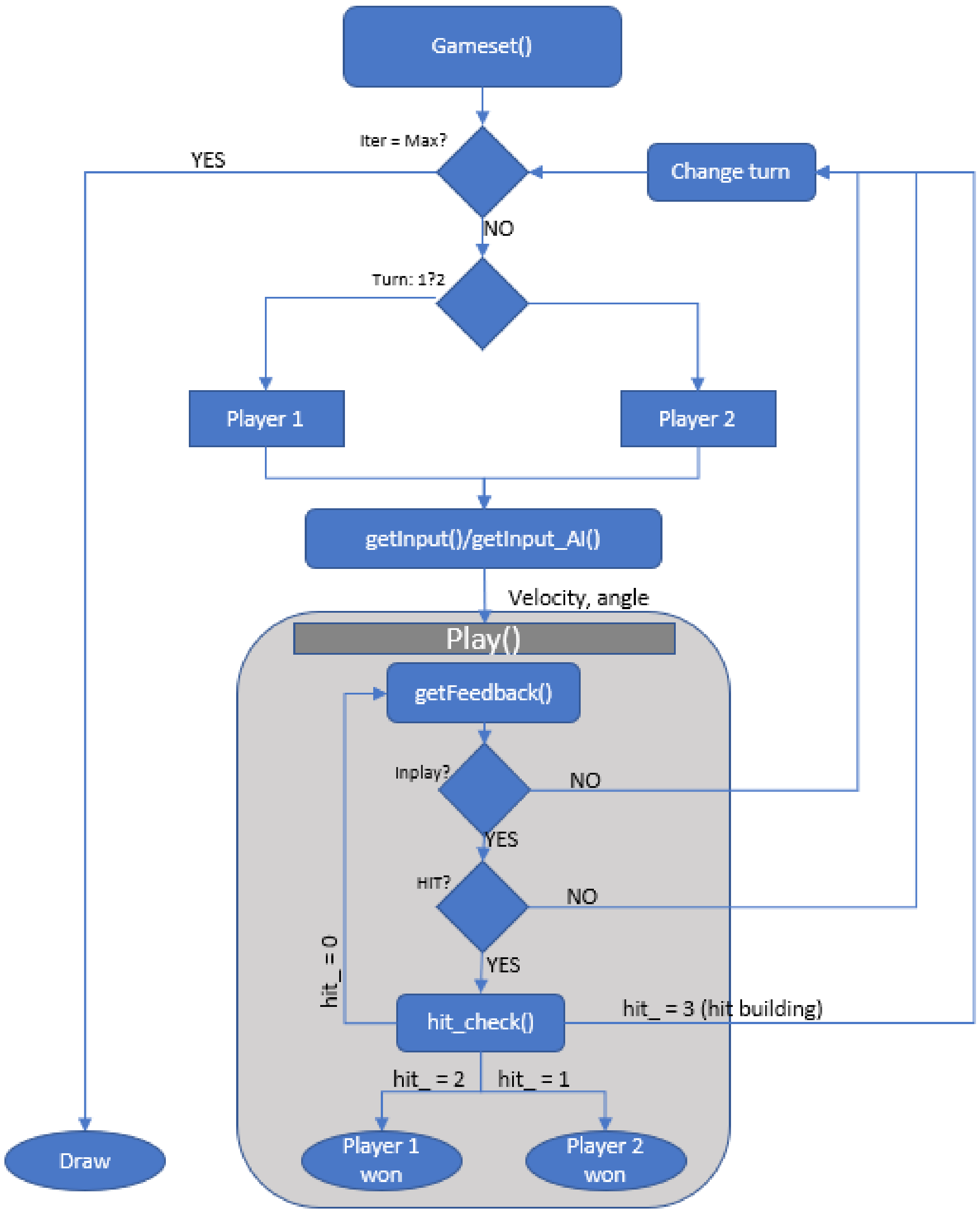


Figure 2: Function diagram of Gorilla game

1.1 Initialization

Initialization feature of the game requires a function that can randomly generate all the required initial values including: positions of two gorilla, heights of buildings, radius of explosion.

- Function *gameSet()*

  We utitilized the *cstdlib* and *ctime* library in order to use methods *rand()* and *srand()*.

  - For positions of two gorillas, it is wanted that two gorillas shoul stands on each side of two halves of the screen. This means that the position of each gorilla should be randomly generated in a specific range. Given the screen size 640x360, it is chosen that gorilla 1 stands within [0,319] and gorilla 2 stands within [320,639].
  - Similarly for heights of buildings, one would always desire heights should be only covering enough the screen so that the playing scenario is visible. We decided the number of buildings is 10 with same width of 64, random height ranging between [50,150].
  - The radius of explosion is manualy fixed as value of 5.

  If one only uses *rand()* method to generate random number, he/she may get the same number every time. In order to avoid this problem, one should additionally use *srand()* method along with *time()* method. By this way, the *srand()* method is called with current time, consequently randomly generated numbers are different everytime.

1.2 Manual game play

(a) Player's inputs

- Function *getInput()*
  - The basic *iostream* library allows users to input value into the game using *cin()* method.
  - However, the expected values are numeric, particularly double type. The funcion needs to be able to eliminate the invalid values which has type different from double. In order to do so, we include a while loop that constantly check the type of input, giving out error messsage when an invalid input comes up, and encourage user to input the valid value.

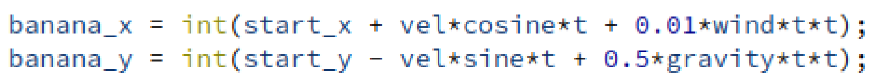
(b) Update state

The state of the game consists of values: positiions of shooting player, positions of targeted player, current position of banana, value of wind.

- Function play*()*
  - The function has parameters consisting of initial values, input values, display structure, and turn
  - Before running the main function and update the states, it would go through a check whether the banana has still been in the screen play, and whether it has hit

anything (gorillas, buildings) via function *bool inplay()* and boolean value *HIT.*

- Next, while it has not hit anything, and still in the screen, the states are computed and updated by function *getFeedback().* The *getFeedback()* requires initial values for gorillas' position, input values, turn, and traveling time. It will return the position of banana every 0.15 s. Details of computation of banana's position is defined as following:



```
banana_x = int(start_x + vel*cosine*t + 0.01*wind*t*t);
banana_y = int(start_y - vel*sine*t + 0.5*gravity*t*t);
```

- The wind value is randomly generated with range [-2,2] and a coeffiecient is assigned to the effect of wind.
- After the banana's position is updated from the *getFeedback()* function, a function *hit_check()* will check if the banana has hit a gorilla, building(s), or nothing.

  +) check the relative distance to gorilla 1, if it is smaller than radius of explosion, return 1.

  +) check the relative distance to gorilla 2, if it is smaller than radius of explosion, return 2.

  +) check the relative y distance to building at that x position, if it is smaller than radius of explosion, return 3.

  +) otherwise, return 0.

(c) Result

- After getting value returned from *hit_check()* function, we will be able to tell who is the winner of the game with value 1 and 2.
- For value 0 from *hit_check(),* it goes back to loop until getting other than 0 value or went out of the screen.
- For value 3 from *hit_check(),* the banana hit the building(s) and do explosion. However, there might be cases where the explosion also damages and kill the gorilla near by. Therefore, we also check whether it hit gorilla when doing explosion on the building.
- There is a maximum value of turns that two players can play for one game. When this values is reached, the result is considered as a draw.

(d) Send to display

- After initialization step is done, the initial values will be sent to Python by function *sendToGUI()* of the struct *initMsg.*

- As soon as the states are updated and *hit_check()* has been run, the states are sent to Python for display by *sendToGUI()* function of struct *displayMsg*

1.3 AI player

The main structure of the program remains the same except the method input values are imported. Instead of manually input values by keyboard after observing states on the screen, an AI player will observe states, give decision what to do and how to do in order to win the game. Only the *getInput_AI()* replaces the *getInput()* function in the program. The lgorithm used for this AI player is decision tree. The main idea of decision tree is that it constructs a graph structure consisting of nodes:

- Decision Node(s): a choice to take between 2 alternatives based on condition.
- End Node(s): an action to take representing the final decision.

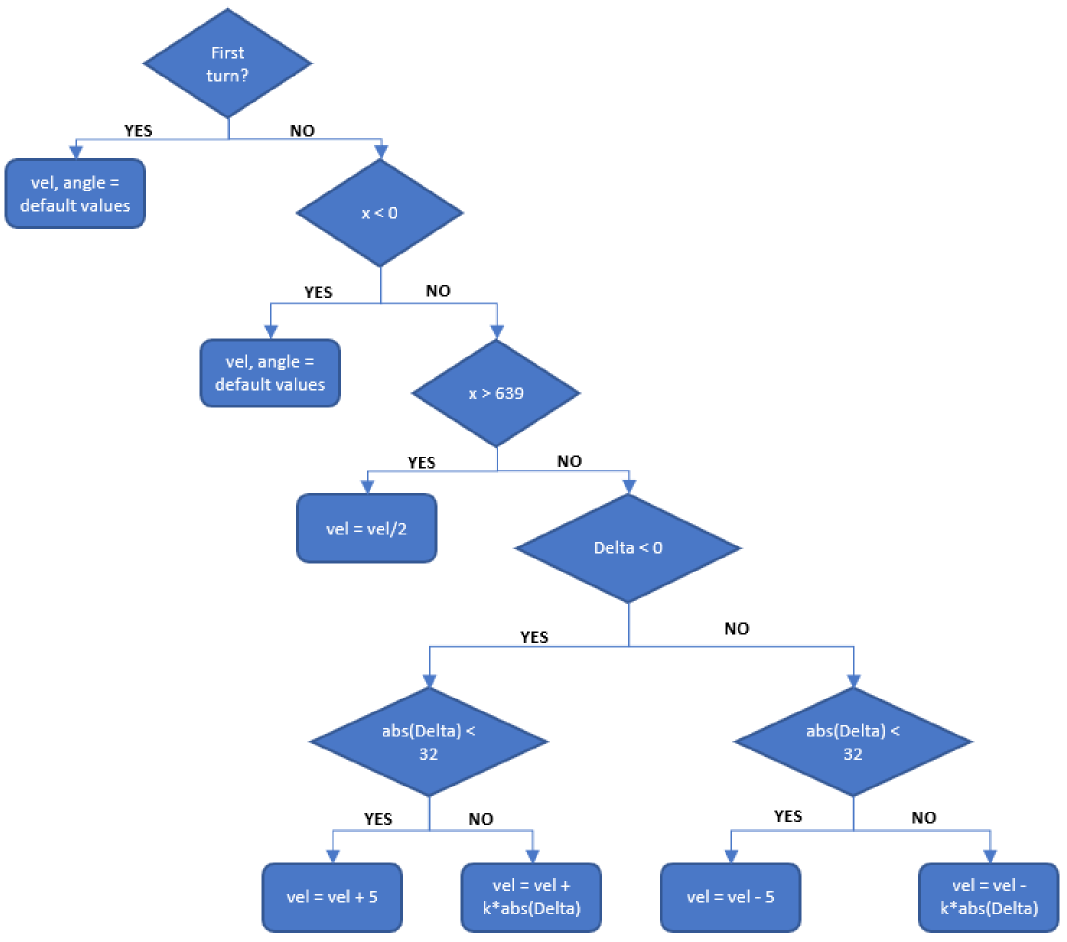


Figure 3: Decision Tree for AI player of Gorilla game

- In this AI algorithm, the AI player is set to be player 1. Considering all possible usecase, the algorithm gives decision of velocity and angle.

- At the first turn, the AI player (as the player 1) knows that the enemy is on the right hand side. Therefore, it is logical to check the right next building by setting default velocity and angle so that it hit the right next building. The default values are 50 for velocity and 85 degree of angle. The value of angle is also remained fixed in all the steps of algorithm. Two reasons for fixing angle: first, since the heights of buildings are random, therefore, 85 degree will always guarantee that it will not hit the right next to building and that it will capabable of hitting the enemy even when the enemy locates between two high buildings (illustrated in the figure below). The second reason is that contrainting angle makes the decision making procedure simpler and easier.

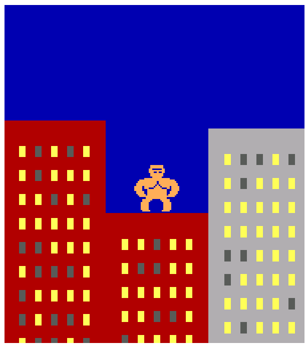


Figure 4: Gorilla "trapped" between 2 high buildings

- Usecases such as the shot went out of screen are also considered and sufficiently dealt with.
- To converge the shot to the position of enemy, firstly it check the relative distance Delta and adjust the velocity according to Delta. And as the shot gets with in the range +/- 32 from the position of the enemy, velocity will be incrementally adjusted by step of 5.

2. Graphical user interface

2.1 Game components design

2.1.1 What is Pygame surface

- Pygame surface is a image object which has a fixed resolution and pixel format. Surface can represent any images, it can be created by calling *pygame.Surface()* function. The only required argument is the width and height of the surface. If no other arguments passed to the function, it would return a black surface. Surface can be taken as very basic components in Pygame. Whatever the images we want to manipulate, the first step would always transfer them to surfaces, if we want to display them through Pygame.
- One remarkable feature of Pygame surface is that it is allowed to build surfaces on other surfaces. To do that, we need to create at least two surface

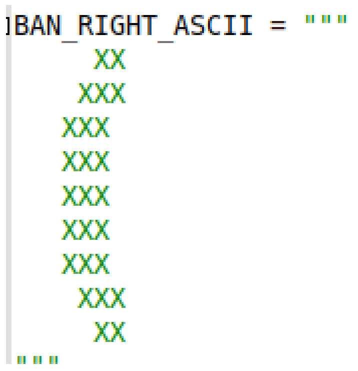
objects in advance, such as Surf1 and Surf2. If we intend to build Surf1 on Surf2, we just need to call *Surf2.blit()* function and pass Surf1 and XY coordinates as arguments to that function, then the Surf1 would be built on Surf2 at the given coordinate. The XY argument is where we want to put Surf1 on Surf2. For every surface in pygame, the origin is always set at the top left vertex, the horizantal axis pointing to right representing X axis and the vertical axis pointing downward representing Y axis, thus the XY argument could be regarded as pinning the origin point of Surf1 at (X,Y) coordiate on Surf2.

- On the other hand, the surfaces are not ready to be displayed unless we build them on a display mode. The display mode is another form of surface for displaying, which can be generated by calling the function *pygame.display.set_mode()*. It is requried to pass the size as an argument to the display mode. The display mode has all the features of surface in pygame. It is allowed to draw other surfaces on the mode. After drawing or building surfaces or shapes on display mode, it won't be shown instantaneously. The *pygame.update()* must be called to show the latest update.

2.1.2 ASCII to surface

- In gengerally speaking, there are two different method to transfer images to Pygame surface. One is to load images by calling the function *pygame.load()* and then transfer the loaded image to a surf. The second is to create game models directly in python and transfer them to Pygame surface. We adopted the second one to build up game models in this project. The basic idea is to create and preserver those models in ASCII code form in python, then define some function to transfer the models from ASCII to surfaces.

The way to generate ASCII code is very easy in python. Here is an example shown below. We can see from the figure below that the banana is composed between """ and """ , the main body of banana is composed of X mark, of course it is acceptable to use different mark as well.



```
BAN_RIGHT_ASCII = """
     XX
    XXX
   XXX
   XXX
   XXX
   XXX
   XXX
    XXX
     XX
"""
```

Once we have created models in python, the other job is to transfer it to a Pygame surface. There is no such a function for us to do the transferration, thus it is necessary to define a function to do the job.

The function _makeSurfaceFromASCII_ shown below is how we adoptted to transfer a ASCII to a surface. This function is referred to another open source Gorilla game by *Al Sweigart*. This fonction works more like creating a new surface as same as the ASCII banana model than transfer the model to a surface.

The first line of the function is to get height by splitting it by '\n', then get the width by counting the maximum X mark number in one line. Up to now we have gotten the width and height for creating a new surface, thus the pygame.Surface() is called with passing width and height as arguments. After the surface has been created, the next step was to pixelate the surface into a pixel array so that we could colorize it pixel by pixel.The final step (written in the for loop) was to detect the position of X mark in the ASCII model and colorize thecorresponding position on the surface.

```
def makeSurfaceFromASCII(ascii, fgColor=(255,255,255), bgColor=(0,0,0)):
        ascii = ascii.split('\n')[1:-1]
        width = max([len(x) for x in ascii])
        height = len(ascii)
        surf = pygame.Surface((width, height))
        surf.fill(bgColor)
        pArr = pygame.PixelArray(surf)
        for y in range(height):
        for x in range(len(ascii[y])):
    if ascii[y][x] == 'X':
            pArr[x][y] = fgColor
        return surf
```

We can generate almost all the static components through this function, except for the buildings and explosion, which will be talked later. The advantage of transfer those models to surfaces in advance is that it allows them to be easily manipulated.

2.1.3 Text object

- A text object is another form of surface in pygame. If we want to display some text information in pygame, the first job is to create a pygame text object. then we can manipulate the object to do what we want. The common way to create a text object is to call *pygame.font* function, it is allowed to specify the font style and font size. Then we need to call the pygame.font.render function to generate a text surface and pass the text content to the function. Font.render function fonctions as creating a surface for text. In other words, it extracts the text content and build them on a rectangle surface which is able to square the text perfectly. The main body in the rectangle is the text, and the rest is background. Text object is allowed to colorize the text and its background. Just like other surfaces, this step doesn't give rise to display the text, we also need to eventually build it on display mode or on other surface to have it able to be displayed. There is one thing should be mentioned that unlike other surface object, whose origin point is always located at top left vertex, the origin of text object can be chosen at center or top left of the text surface by choosing the attribute Topleft or Midtop respectively.

2.1.4 Buildings and explosions

2.1.4.1 Buildings

- The buildings and explosions are built in the way different from other static components such as gorillas or sun. The building height and position real depends on the data passed by C++, and where the explosion happens also depends on where the banana hit the buidings. In other words, it also depends on the data passed by C++. Since they are not fixed as gorilla or sun, there is no way to create and build them in advance. The good news is that both of them are in regular shape, thus we don't create some irregular shape like what we did for gorilla or sun. Here is a segment of code shown below indicating how the buildings are set up

- The building data received in python is in a 1x640 list, in which every element represent the height of the building. Since each building takes 64 pixels(the width) in this project, the list can be divided into 10 different batches, each of them has 64 identical values.
  Since the building information will be randomly generated in c++ for

each turn, therefore there is no way to build up a surf for building before hand. The solution to create buildings is to create a big surf and to draw each building according to the data extracted from the list. For example, since the information extracted from the list only indicating the height of each building, thus we need a line as base for all the buildings, then call draw rectangle function to construct the buildings on the surf. The original point of pygame display mode and all other surfaces are at the top left vertex, so the y coordinate of each building is the y position of the base line subtracts the corresponding height, and x coordinate is always the x coordinate of the previous building plus 64.
For the draw rectangle function, it only needs the coordinate of the top left vertex of each building, and the height and width of each building, there is no need to provide coordinate of each point on the building.

- After drawing all the buildings on the given surface, we need to return the surface, then we could use that surface as a background in the display mode. The advantage of drawing buildings on another surface rather than directly on the display mode is that it is easier to display the dynamic. In displaying the dynamic part, we need to show the flying trajectory and orientation of banana at each time instance, at the same time covering the previous movement. If we draw everything on one surface, there is no way to cover the previous movement.

2.1.4.2 Explosions

- The purpose of doing explosion was to show how the buildings are damaged when hit by banana, thus the explosion can be taken as a filled circle centered at the banana when it hits buildings, which can be done by calling pygame.draw.circle function. See(Figure 6), the circled parts are explosion caused by banana. Naturally, they are circles filled in sky color drawn on the background surface.

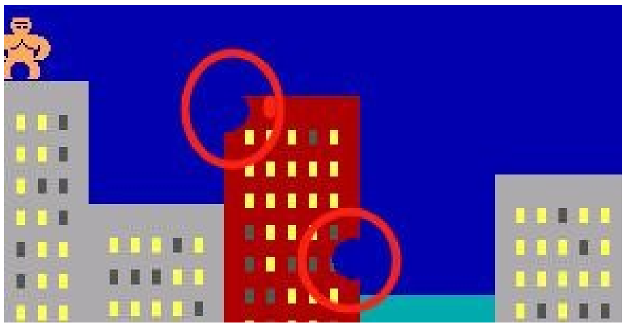


Figure 6: Explosions caused by banana

- The only confusion was that we drew explosion on the same surface as which banana locates primitively, since it is caused by banana. However, the problem was that if we put the explosion on the same surface where banana locates, it will disappear once we update the background, in other words, the explosion will be covered by the background surface. To figure out this problem, the point is that explosions should belong to static part rather than dynamic part. In other word, once the explosion is caused, it remains there and won't change until the end of the game, unlike the banana, whose movement has to be updated at each time instance.

2.2 Dynamic displaying

- The banana is the only dynamic component in the project, its position and orientation has to be updated at each time instance. The method to show the dynamic is to draw the banana on display mode, the display mode can be taken as a surface too, and then covered by the background surface, which is mentioned before. At the next time displaying the banana, its position and orientation will be changed. And we have updated the background surface, the previous banana will never show up again. It is very important to update the background surface after displaying the banana, otherwise the banana at every time instance will be shown on the surface, which is definitely not what we want. see(Figure 7).
- We can see from the figure below that the parabola of banana movement is presented entirely. it is not covered because we didn't update the background surface at each time instance after displaying banana. The solution to this problem is to just update the background surface every time the banana shows up.

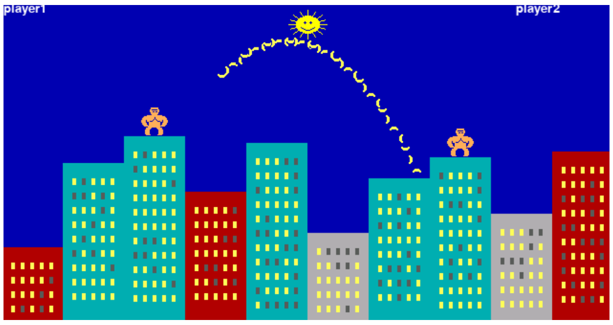


Figure 7: the banana movement without being covered by background

- Since the banana will be spinning in air after throwing a banana to a nother player, it would be better to show the banana changing the pose at every displaying. the method was to create four different poses representing the banana on up, down, right and left orientation in advance. Then generate four different numbers as flag to determine which pose should be displayed. For instance, we can generate 4 integers such as 0, 1, 2, 3, each integer represents a banana pose, then the only job now is to loop the 4 integers. if there is no problem looping the four integer, then there wouldn't be any problem for us to show the four banana poses either.

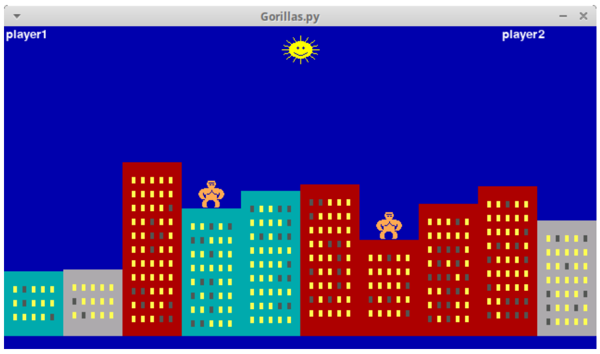
IV. Testing and Evaluation

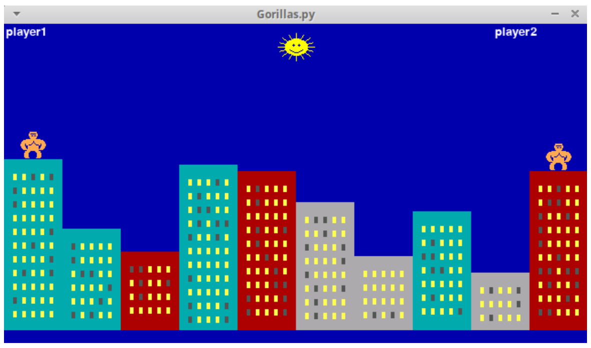
1. Manual play

(a) Initiliazation

The game has been able to generate different scenrio. The GUI successfully display the initial setup receiveed from game's mechanics.

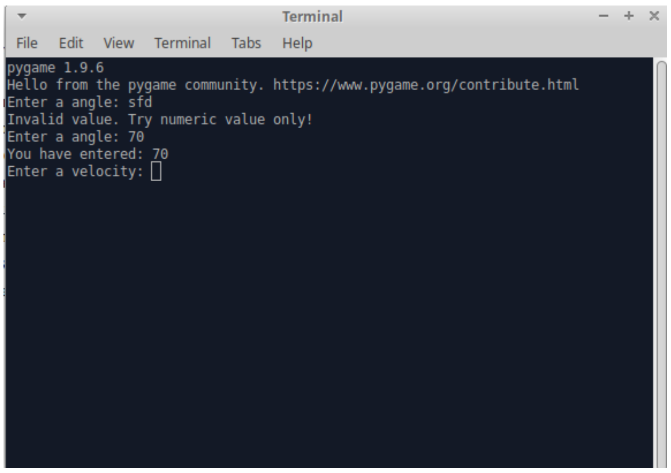
Figure 8: Initialization display of Gorilla game





(b) Input by user

The manual user is able to input from keyboard via terminal.



```
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Enter a angle: sfd
Invalid value. Try numeric value only!
Enter a angle: 70
You have entered: 70
Enter a velocity:
```

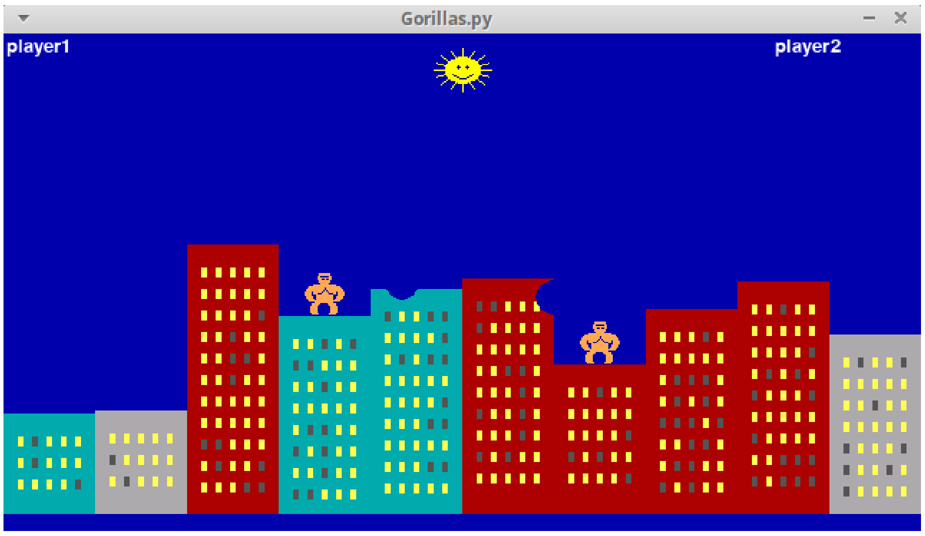
Figure 9: Invalid value message displayed on the terminal

The error message is displayed on the terminal when invalid input is inputted. The user is asked to input only valid value.

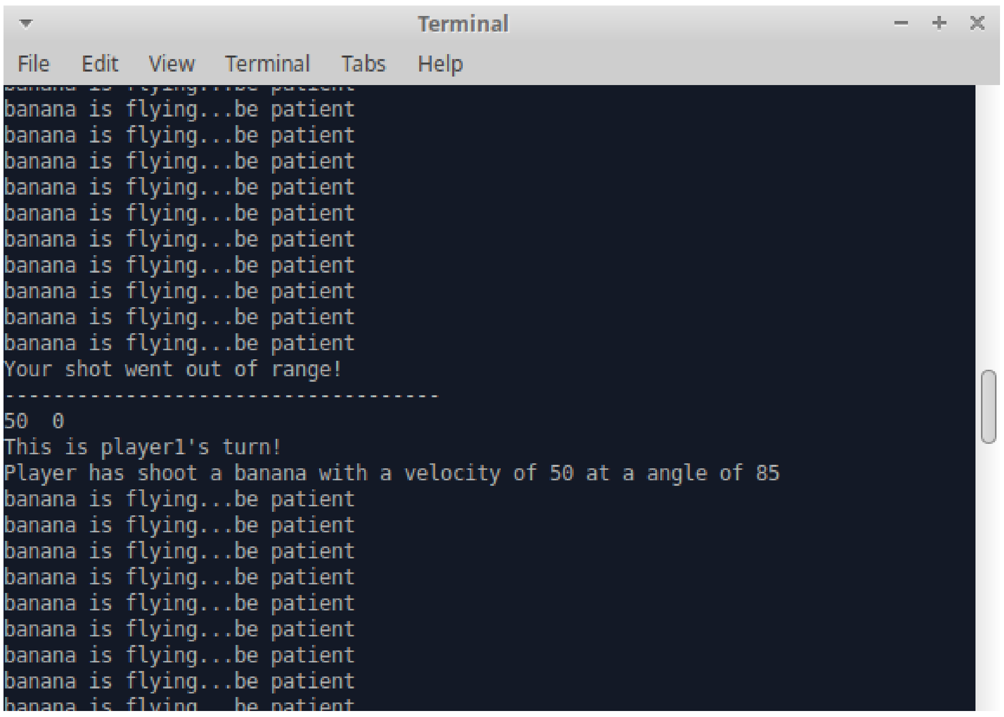
(c) State update

The display shows how the banana shot landed on the building.

Figure 10: States update example when buildings was hit



When the shot went out of the screen, a message is showed up to announce the player.



```
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
Your shot went out of range!
------------------------------------
50  0
This is player1's turn!
Player has shoot a banana with a velocity of 50 at a angle of 85
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
banana is flying...be patient
```

Figure 11: Message shows the shot went out of screen

When a gorilla was hit, the GUI would show the explosion as below.

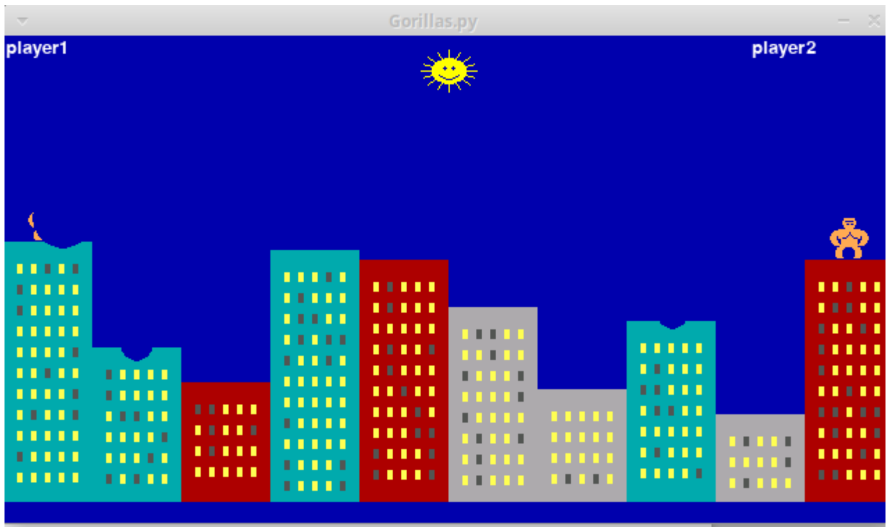
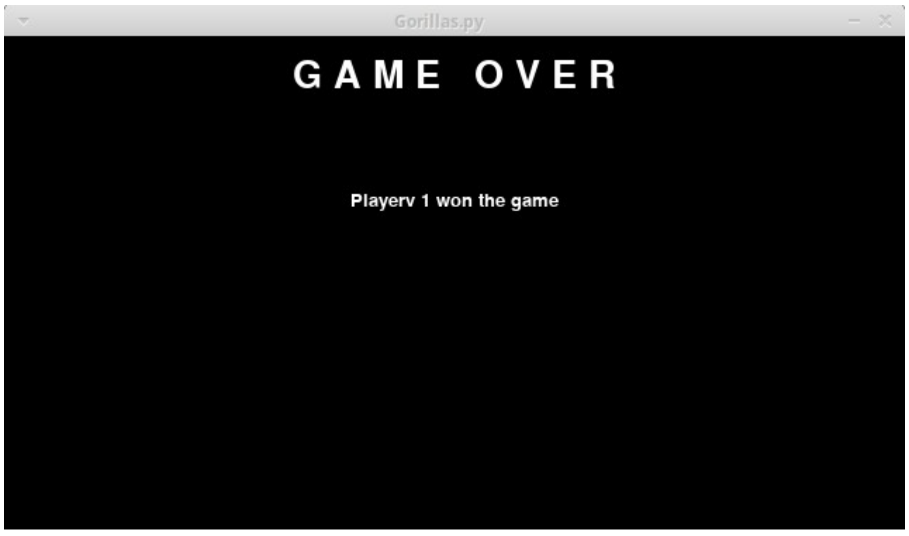


Figure 12: GUI displays when one of the gorillas was hit

(d) Result

The game ended with a winner when one of the gorilla was hit. The GUI shows the final screen as below.



GAME OVER

Playerv 1 won the game

Figure 13: Final screeen example when there is a winner

2. AI vs manual player

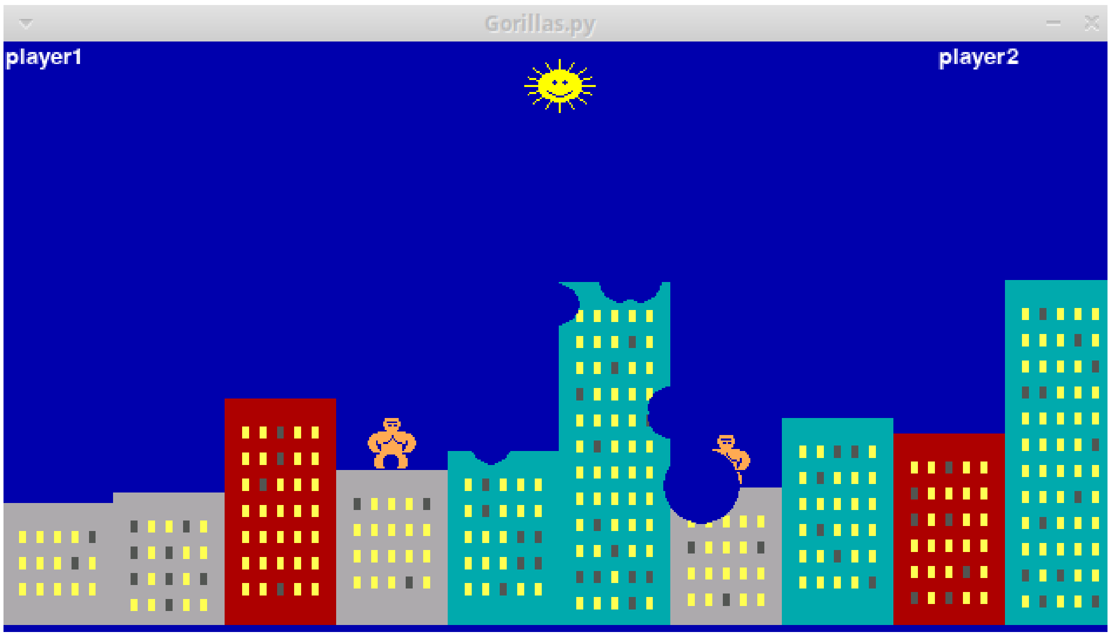


Figure 14: Display of AI player's performance in a difficult scenario

The AI player (player 1) managed to win the game with very dificult scenario, where the enemy "hides" behind a very high building. This proved the efficiency of the AI algorithm even with challenging scenario.

V. Conclusion

In this project, we have successfully developed the Gorilla game using C++ and Python/Pygame. The game has proved that it satisfied all the goals of the project. Initialization of the game allows players to have different scenario every game. The manual input and AI input feature functioned very well. Game's states are updated and showed on the GUI without delay or misinformation. The game allows to play in different mode: 2 manual players, 1 manual player vs 1 AI player. It also ensures the game always has a final result at every game to show on the GUI.

For further development, the game can be used as a platform for learning AI programming. In addition, sound effect can be added to the game to increase the enjoy level of player. In term of improvement, code can be optimized to reduce running time and make the game faster.

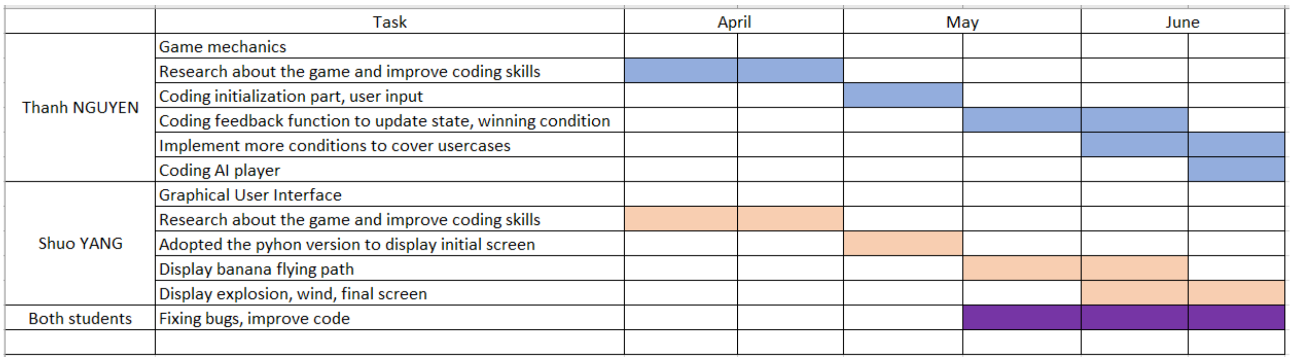
VI. Reference

1. The Gorilla game written in python: http://inventwithpython.com/

2. AI programming for game: https://www.gamedev.net/tutorials/programming/artificial-intelligence/the-total-beginners-guide-to-game-ai-r4942/

3. Pygame website: https://www.pygame.org/docs/

Appendix : Individual Project Log



	Task	April		May		June	
Thanh NGUYEN	Game mechanics						
	Research about the game and improve coding skills	🟦	🟦				
	Coding initialization part, user input			🟦			
	Coding feedback function to update state, winning condition				🟦	🟦	
	Implement more conditions to cover usercases					🟦	🟦
	Coding AI player						🟦
Shuo YANG	Graphical User Interface						
	Research about the game and improve coding skills	🟧	🟧				
	Adopted the pyhon version to display initial screen			🟧			
	Display banana flying path				🟧	🟧	
	Display explosion, wind, final screen					🟧	🟧
Both students	Fixing bugs, improve code				🟪	🟪	🟪