PD-110 Group 21 Final Project Report: The Professor Jump Game

B10705003 CHEN, YEN-SHAN, Department of Information Management, NTU
B10705004 LIN JHIH-YU, Department of Information Management, NTU
B10705005 CHEN, SZU-JU, Department of Information Management, NTU
B10705011 LIN, YU-JIE, Department of Information Management, NTU
B10705033 CHANG, HE-CHIA, Department of Information Management, NTU

1 OVERVIEW

PROFESSOR JUMP is a game extended from the Doodle Jump, but we change the characters into our professors and add some interesting functions and tricks to it. We hope this game can not only enhance our programming ability but also bring joy to every player.

2 INTRODUCTION

There are three levels in the PROFESSOR JUMP game, each featuring a distinct character (more specifically, a professor in the Department of Information Management). In general, one should play longer and jump higher to get higher points. Other rewards and penalties will be introduced in the following section. The game will record a player's highest score until he/she closes the window, so one could try as many times as he/she wants to get the highest rank!

3 FUNCTIONS

3.1 Game Setup and Basic Functions

3.1.1 Three Levels. As mentioned, there are three levels in the game, and each level has its own character (as shown in Fig.1.) and background. The player starts in Level 1. Once he/she reaches the 200-point threshold, the player switches to Level 2. When 500 points is gained, the player gets to Level 3, which is also the highest level in the game. Information about the levels is listed in Table 1.

Fig. 1. Characters in the Game

Table 1 Characters and Background for Each Level

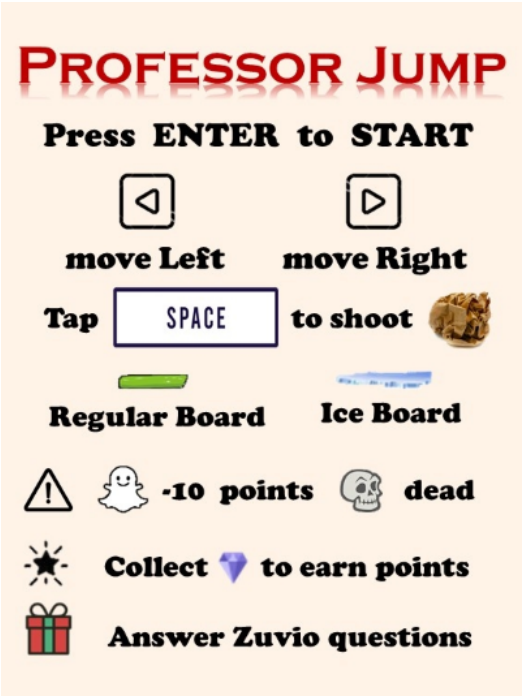
Level	Character	Background (Theme)
1	Professor Ling-Chieh Kung	Underwater World
2	Professor Yuh-Jzer Joung	Land Park (In the Desert)
3	Professor Chien-Chin Chen	Outer Space

3.1.2 Platforms. There are two types of platforms in the game. There is a regular platform (a green one), which can be jumped on repeatedly. The ice platform (the blue one with an icy texture), on the other hand, can only be jumped on once and will disappear immediately once jumped on. A character jumps up and down automatically, and the player uses the left and right key to control which specific platform the character lands on. For both types of platforms, a player collects 5 points for each jump.

3.2 Additional Functions

There are 7 additional functions in this game, which either (1) makes the game more entertaining and enjoyable or (2) provides the player with some rewards or penalty. Fig. 2. is the starting menu of the game, in which we introduce the basic rules and scoring methods.

Fig. 2. The starting menu of the PROFESSOR JUMP game



3.2.1 Ghosts (Penalty). The ghost falls from the top of the window to the bottom of the window every once in a while. If the character touches the ghost, 10 points will be deducted.

3.2.2 Skeletons (Penalty). Similar to the ghost, the skeleton also falls from the top of the window. However, the skeleton appears less frequently and thus results in a more severe penalty when touched: Game over directly.

3.2.3 Bullets (To Avoid Penalties). Given the ghosts and skeletons that can attack our players, we equip our characters with (unlimited) bullets for defense. When a ghost or skeleton is shot by a bullet, it dies and disappears. To shoot a bullet, the player should press the space key.

3.2.4 Diamond (Reward). Diamonds appear randomly on top of platforms. If a player jumps on a platform with a diamond on it, it collects the diamond and earn points. Since the diamond appears extremely frequently (about 25%-30%), the player can only receive a 1-point reward per diamond.

3.2.5 Gift Boxes (Hopefully a Reward). Gift boxes are also generated randomly on top of platforms. When a player gets a gift box, a "Zuvio Question", which is a chance of earning or losing points, will earned.

3.2.6 Zuvio Questions (Penalty or Reward). Zuvio Questions are the questions in the gift boxes. They are all multiple choice questions related to "NTU", "Department of Information Management", or "Programming Design Class". The player should answer the questions with the mouse. A correct answer will earn the character 20 points, while a wrong answer will result in a 20-point deduction from the score.

3.2.7 Sound Effects (Entertaining). To make the game more entertaining, we have added various sound effects to the game. They are played, for example, when the character jumps, answers questions correctly or wrongly, shoots a bullet, etc.

3.3 Scoring Method

Table 2. summarizes of all the possible ways to earn or lose points. Note that in this scoring method, it is possible that a player transfers from a higher level to a lower level.

Table 2 Possible Ways to Earn or Lose Points

Function	Penalty/Reward	Points
Jump on Platforms	Reward	+5
Touch Ghosts	Penalty	-10
Touch Skeletons	Penalty	game over
Touch Diamond	Reward	+1
Touch Gift Box	Reward	a question
Zuvio Question	Penalty/Reward	+20 if answered correctly -20 otherwise

4 ALGORITHM

We will use three subsections to describe the design of our algorithm. First, we focus on the basic, and also the most fundamental part of the PROFESSOR JUMP game, which is to generate the platforms randomly and to make them move and disappear. Then, we introduce how the skeletons and ghosts were able to fall from the sky and how the diamonds and gift boxes were generated and collected.

4.1 Platforms

4.1.1 Initialization. The initial state of this game is to place several platforms randomly in the window. To achieve this goal, we first create an array of n PLATFORMs, where n is the total number of platforms. A PLATFORM (the capitalization indicates that it is a self defined data type) is a struct with two attributes, position x and position y. In the beginning of the game, we generate the position of each PLATFORM using the rand() function.

After testing, we have found that 12 is the best value for n - a big n results in platforms overlapping; a small n makes the character die too easily.

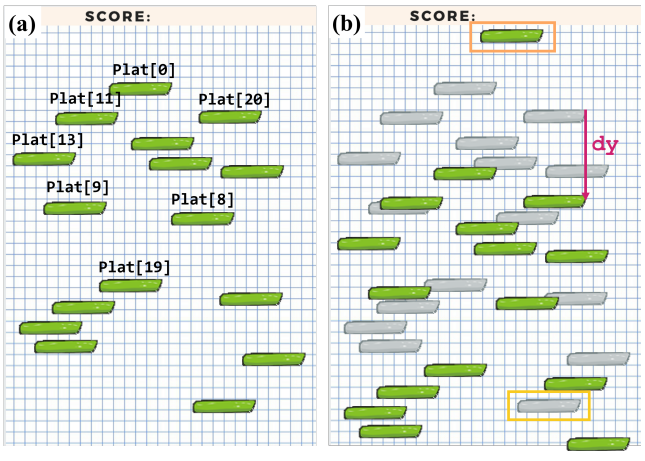
4.1.2 Move. After generating the initial position of each PLATFORM, our next task is to let the platform move continuously (like an animation). Our approach is to create a variable dy. In each iteration (before the game ends), we add dy to the y position of each PLATFORM. By adding a very small number (after testing, 0.2) to each PLATFORM, we get the effect of all platforms moving continuously.
Another point worth mentioning is that we want to create a "bouncing" effect. Therefore we temporarily change dy to 10 whenever a character jumps on a platform.

4.1.3 Deletion and Renew. Finally, when a platform moves out of the window, we generate its new position at the top of the window and start a new cycle.

Note that in this algorithm, the amount of platforms in the window at each moment is constantly 12 (because a deleted platform is again generated immediately).

With Fig. 3., we once again explain the implementation of the algorithms for the platforms. Fig. 3. (a) shows that the platforms are stored in an array (named Plat), and that they are generated randomly in the beginning of the game. Fig. 3. (b) shows how platforms are moved and renewed. The gray platforms indicate the platform positions before an iteration. After an iteration, a small number (dy, the red arrow in Fig. 3. (b)) is added to each platform, and the green platforms reveal the new platform positions. Finally, if a platform falls out of the window (the one with a yellow frame), its position is immediately regenerated with the rand() function and placed at the top of the window (the one with an orange frame).

Fig. 3. Diagram to Show How Platforms are Initialized and Moved



We use the psuedocode on the next page to summarize the three parts of the algorithms for platforms: Initialization, Move, and Renew.

Algorithm Platform Algorithm: Initialization, Move, and Renew

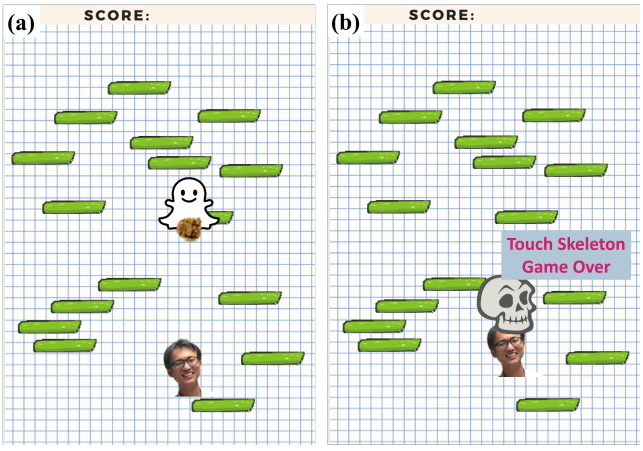
```
//initialization begins
for each platform do
    Generate its x and y position
end for
while the game is running do
    //Start moving
    for each platform do
        Add dy to its y position
        if the platform moves out of the window then
            //Start renewing position
            Generate its new x position with rand()
            Generate its new y position = 35 ▷ the score bar width
        end if
    end for
end while
```

4.2 Ghosts and Skeletons

The appearance of skeletons and ghosts follow a similar logic. However, since we will not have two ghosts or two skeletons fall down at the same time, we do not need an array to store GHOSTs or SKELETONs. Instead, we have only one ghost and one skeleton, and both of them are structs with two attributes: position x and y.

4.2.1 Appearance. We want to let ghosts and skeletons fall from a random position every once in a while. We create two counters K and J, and both are increased in each iteration. A ghost will fall down from the top of the window every 100 iterations and a skeleton will fall down every 1250 iterations. After they start falling, $dy = 3$ is added to their y positions in each iteration. The positions of the ghost and the skeleton are not reset until the next time they start to fall. Note that a skeleton falls less frequently than a ghost because it is associated with a more severe penalty.

Fig. 4. Additional Functions for the Ghost and the Skeleton



4.2.2 Touch. The ghost and the skeleton have 2 other additional functions. First, when they are shot by bullets, they should disappear (as shown in Fig. 4. (a)). Second, when our character touches the ghost, 10 points will be deducted; when it touches the skeleton, the game ends directly (as shown in Fig. 4. (b)). To achieve this goal, we compare the positions of our character, the ghost and skeleton, and the bullets in each iteration.

Algorithm Ghost and Skeleton Algorithm

```
K = rand(1, 100) and J = rand(1, 1250)
while the game is running do
    K++ and J++
    //dropping begins
    if K%100 == 0 then
        set ghost position (rand(0, 400), 35)
      ▷ 400 is the window width and 35 is score bar width
        drop down a ghost
    end if
    if J%1250 == 0 then
        set skeleton position (rand(0, 400), 35)
        drop down a skeleton
    end if
    Add 3 to ghost and skeleton y position
    //judging touch begins
    if the positions of the ghost and character overlap then
        Deduct 10 points from the score
        Show text to indicate score deduction for 0.7 seconds
    end if
    if the positions of the skeleton and character overlap then
        Show text to indicate player dies for 0.7 seconds
        break
    end if
    if a bullet touches the skeleton or the ghost then
        set ghost position to (1000, 1000)
                              ▷ somewhere out of the window
    end if
end while
```

4.3 Diamonds and Gift Boxes

The algorithms of gift boxes and diamonds also resemble that of the ghosts and skeletons. The only difference is that the gifts and diamonds are associated with the array of PLATFORMs (i.e., the existence of a diamond or gift can be considered as an attribute of the PLATFORMs).

4.3.1 Appearance of Diamonds and Gift Boxes. In order that the diamonds and gifts appear every once in a while, we make use of the index of the array of PLATFORMs. If the index of the PLATFORM is multiple of 3, we place a diamond. If the index is a multiple of 8, we place a gift.

4.3.2 Touch and Disappear. To easily judge whether a gift or diamond is touched, we make use of an already-written function of the PLATFORM: When a character touches a platform, we create a bouncing effect (please refer to Section 4.1.2). We add in this function that, if the touched platform has a corresponding object (a diamond or a gift box), we let the object disappear and trigger an appropriate reaction (+1 point or answer Zuvio Question).

4.3.3 Zuvio Questions. First, for each Zuvio Question in the Question Bank, we intentionally set their filenames to be "Q"+index+".png" so that they can be read into an array QUESTIONS with a for loop.

Then, we set a variable "whichQues = 1" to record which the next question should be. When a character collects a gift box, it answers the Zuvio Question with the index number "whichQues". 1 will be added to the variable "whichQues" whenever a question is answered. Note that under this algorithm, the Zuvio Question is generated "when the character collects the gift" instead of "when the gift first appears".

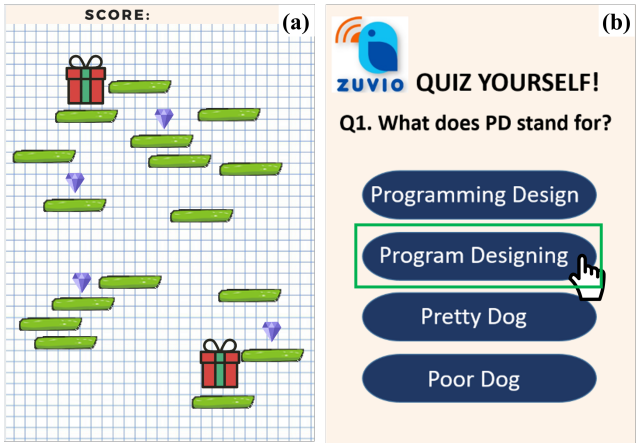
After the Zuvio Question appears, we wait until a mouse click is detected. Upon detected, we instantly capture the position of the mouse and analyze which option the player has chosen. A green frame (as shown in Fig. 5. (b)) will be displayed to show the player's choice and depending on whether the player has chosen the correct answer, there will be some sound effects and points adjustments.

Algorithm Diamonds, Gift Boxes and Zuvio Questions Algorithm

```
for i in range (total number of questions) do
    load file ("Q"+question index+".png") into questions[i]
end for
new variable whichQues = 1
while the game is running do
    //appearance
    for each platform do
        if index of platform %3 == 0 then
            set diamond position on top of Plat[i]
            show diamond
        else if index of platform %8 == 0 then
            set gift box position on top of Plat[i]
            show gift box
        end if
    end for
    for each platform do
        if the platform is touched then
            //bounce...
            if the touched platform has a diamond on it then
                get rid of the diamond using flags
                +1 point to game score
            else if the touched platform has a gift box on it then
                generate Zuvio Question
                capture mouse click position
                analyze which option was chosen by player
                if the answer is correct then
                    play correct sound effect (Congratulations!)
                    +20 points to game score
                else
                    play wrong sound effect
                    -20 points to game score
                end if
            end if
        end if
    end for
end while
```

The psuedocode below summarizes the algorithms for diamonds, gift boxes, and Zuvio Questions. Fig. 5. shows how diamonds and gift boxes are displayed in the window and what the Zuvio Question interface looks like.

Fig. 5. (a)How Diamonds and Gift Boxes are Displayed and (b) Zuvio Question Interface

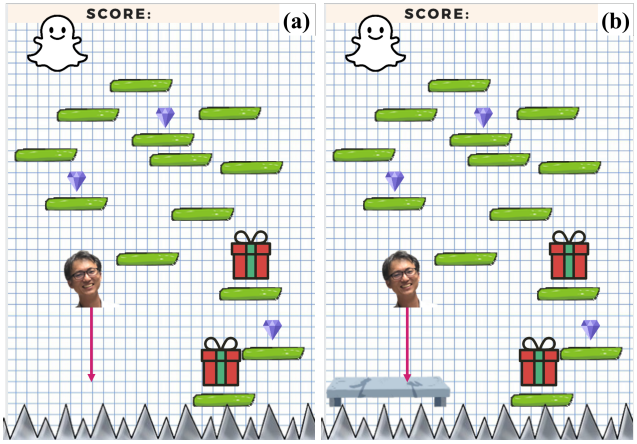


5 PROBLEM AND SOLVING

One major problem that occurred was that a bad randomization when initializing the positions of the array of platforms made our character die very easily (as seen in Fig. 6. (a)).

Our approach to this problem is to set a starting platform (as seen in bottom-left corner of Fig. 6. (b)) in the window at the beginning of the game so that the character can survive the first jump even if randomization for platform initialization wasn't good.

Fig. 6. (a)A Bad Randomization for the Initialization of Platforms and (b) Starting Platform Solution



6 FUTURE WORK

There is still room for improvement in our final project and the following are some new functions we want to advance and create in the future.

6.1 Different Language Versions

The player can choose either the English or Chinese version for both the user interface and the Zuvio questions.

6.2 New Levels

Create more levels and include new characters such as the principal or TAs.

6.3 New Platforms

If the character jumps on the yellow platform, it may jump 2 times higher than jumping on the normal platform. Another idea is that if the character jumps on a black platform, which rarely appears, the character will have to go back to the previous level.

6.4 Auto-aiming System

The character can aim the ghost or the skeleton from different angles and use the bullet to attack them to protect itself.

6.5 Number of Bullets

The number of bullets will be limited and be refilled by touching the diamond or answering the Zuvio questions correctly.

6.6 HP Value

When the character touches the ghost, the HP value will be minus instead of the score. The character should touch the first aid kit, which is still work in progress, to recover. It's game over once the HP value becomes zero.

7 WORK DIVISION

The whole team collaboratively finished all parts of the project, including brainstorming, coding, and preparing for the presentation. For the coding part, after everyone had successfully installed SFML and a first version of our game was completed, we held weekly discussions to (1) share and combine our newly written codes, (2) report newly discovered bugs, and (3) brainstorm new functions that would make the game more interesting. Then each member on the team picks some work to accomplish during the week according to her strengths. Although each member coded separately, we stayed in close contact via a group chat and were able to discuss about and solve each others' problems timely. In addition, new thoughts, resources, update versions of codes were also exchanged efficiently. For detailed group division, please refer to the list below:

7.1 Chen, Yen-Shan's Responsibility

Coding: platform-related functions, ghost, skeleton, diamond, gift box, and Zuvio Question function
Other: Zuvio Question Bank (design), PowerPoint for group presentation (only algorithm part), and organizing and formatting paper report

7.2 Lin Jhih-Yu's Responsibility

Coding: sound effects, edge control (so objects don't fly out of the window), conditional statement to judge whether it's game over
Other: SFML install guidance

7.3 Chen, Szu-Ju's Responsibility

Coding: start and end conditional statements, shooting function, score counting, organize code (make the code more readable to everyone and make zip file)
Other: start and end, level up interface design, and font choice

7.4 Lin, Yu-Jie's Responsibility

Coding: parameter adjustments, platform-related functions
Other: character design, PowerPoint design for group presentation, Zuvio Question Bank (contents)

7.5 Chang, He-chia's Responsibility

Coding: sound effects, level-up function
Other: background pictures, sound effect resources

8 REFLECTIONS

8.1 Chen, Yen-Shan's thoughts

I would use "cooperative", "efficient", and "happy" to describe our team. We always worked in a joyful atmosphere, and though we didn't give each other pressure, everyone was willing to fully contribute to the project and realize the creative thoughts proposed by our groupmates. Moreover, our group was extremely efficient because we divided the work according to individual strengths. In addition, we were always closely in contact, and this allowed people to solve/ discussed about their problems promptly.

For me, the biggest problem I encountered while working on the final project was that I wasn't able to successfully link the sfml libraries to the IDE I used - CodeBlocks. Following the online tutorials didn't seem to work out for me. After struggling for a long time, I decided to take my groupmates' advice, and installed Dev C++ instead.

The most interesting part was that I was able to use a similar logic to complete many seemingly unrelated functions of the game. For example, though the ghost seemed to have nothing to do with the platforms, the codes for the two objects were highly similar: (1) The ghosts just moved (relatively) faster than the platforms, and (2) The conditions to judge whether our character touched the ghosts or the platforms were actually exactly the same. Functions for the skeletons, diamonds, and gifts also followed a similar logic. This, seeing a set of codes to have many applications, was very fascinating to me. At the same time, I have found that modular programming can be very helpful in such scenarios.

8.2 Lin Jhih-Yu's thoughts

First of all, I'm really grateful to be in a such cooperative team. Compared to the midterm project, when most of the people are first time working as a team, we discussed and divided the work more efficiently. We all made our best effort to create the game. Besides the soft skill I learned from teamwork, I have developed the skill of checking the reference and finding the applicable functions for our project, which is also essential for programming. Though the result is not exactly perfect, I am still proud of our work and also treasure what I learn from the experience.

8.3 Chen, Szu-Ju's thoughts

This Profesor Jump project means a big difference to me. It's kind of the first time we create a game from nothing. In this project, I also learned a lot, like how to use the SFML tool by myself and how to understand what each code stands for, which is an essential ability in programming. It's a pleasure to cooperate with this group, everyone does her best to come up with different ideas to make the game more and more interesting. Also, we all try our best to improve the game functions but there is still a lot of space for us to work on due to the time pressure. I'm very glad that I have this experience with this group and learn some programming techniques from others. It literally is the best and most interesting stuff in this class!

8.4 Lin, Yu-Jie's thoughts

Different from the midterm project, the final presentation project was much more interesting. I joined a new group, met some different team members, and had a better time. The thoughts of the game we designed came up from our one-semester life in NTU. The whole process was very efficient and harmonious. We only spent less than two weeks producing the game, including coding the main functions, coming up with so many ideas, and twelve Zuvio questions. The most important thing to me in the final project is that I finally had the ability to "do something" rather than just watching other members and don't know what to do just because of my lack of ability. I think this is definitely the most meaningful thing in this semester.

8.5 Chang, He-chia's thoughts

This final project not only gave me a chance to learn how to use some tools, such as SFML, to design our own game but also taught me the importance of teamwork. During the discussion, every team member does their best to brainstorm many interesting functions to make our game more entertaining. Although there is still room for improvement, I'm having fun and learning something from it, and I also find myself making progress.

9 RESOURCES

For further references of the source code and all materials (including fonts, audios, and images) click here, go to the next url: https://reurl.cc/Opkx89, or scan the QR code below.

Fig. 7. QR code for source code and materials