The Development and Implementation of a 2-Dimensional Platform Adventure Game

Ruochen Ding [*a+], Yancheng Su [b+], Yifei Xia [c+]
*Corresponding author email: 13807180551@163.com

[a] School of Computer Science, Wuhan University, Wuhan, 430072, China, 13807180551@163.com

[b] College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410000, China, 964800156@qq.com

[c] School of Computer Science, Wuhan University, Wuhan, 430072, China, 974196263@qq.com

+These authors contributed equally to this work and should be considered co-first authors.

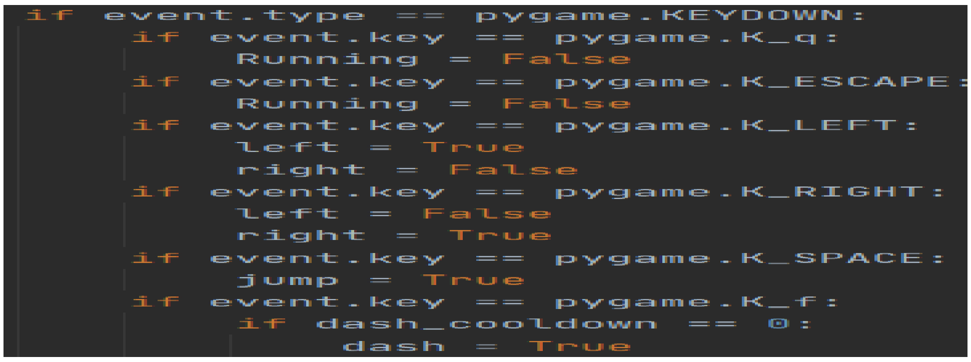
Abstract: The three authors from the same python game programming course have made a prototype of a 2D Platform Adventure Game. The game is mainly about a hero controlled by player fights against monsters. Though it seems to be rough as a game, they successfully applied what they have known about python and Pygame pack into programming. This article is about the algorithms and designs, including the basic moving, jumping, and specially designed dashing and attacking actions in this game prototype, using Pygame and its various pixel-based functions. By the time this article had finished, this project is still can't be regarded as a real game, and the authors will probably add more features into the game.

Keywords: Python, Pygame, Platformer, 2D game

1 INTRODUCTION

Pygame is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language. It is highly portable and runs on nearly every platform and operating system [1]. In this game project, the authors develop their game and implement some algorithms of gaming on the basis of Pygame and Python. This game is a 2D platform adventure game, a kind of game that all roles and subjects are on a 2D level. The player-controlled hero needs to fight against different kinds of monsters and struggle to survive to the end. As shown in Fig 1, there are many basic actions available for players, including running to the left and right, jumping, attacking to the enemies in the direction of the face. Also, "Dash" is added into the game as another function for the hero. Generally, hero can dash for several frames, which keeps him from being hurt by enemies. More designs about attacking and dashing are introduced in Cp. 4: "Methodology: Numerical Design in Game".

ICICA 2022, December 02-04, Chongqing, People's Republic of China
Copyright © 2023 EAI
DOI 10.4108/eai.2-12-2022.2328047

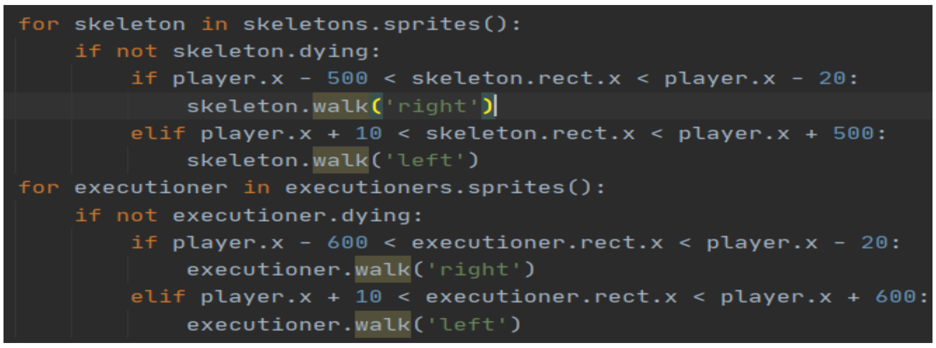

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_q:
        Running = False
    if event.key == pygame.K_ESCAPE:
        Running = False
    if event.key == pygame.K_LEFT:
        left = True
        right = False
    if event.key == pygame.K_RIGHT:
        left = False
        right = True
    if event.key == pygame.K_SPACE:
        jump = True
    if event.key == pygame.K_f:
        if dash_cooldown == 0:
            dash = True
```
Fig 1. Keys to control the action of hero.

2 METHODOLOGY: POSITION DETECTING ALGORITHM

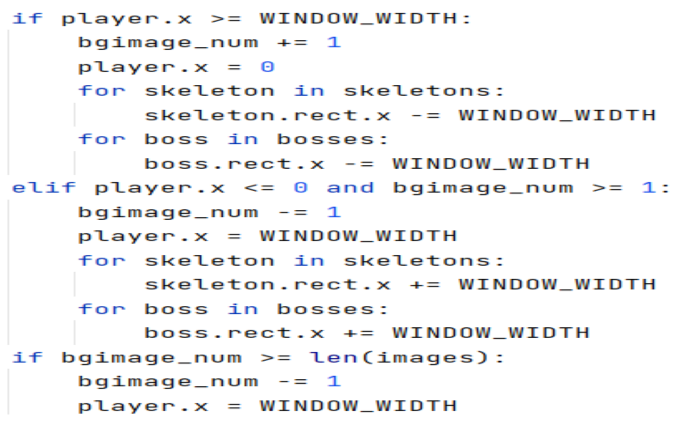
Here comes the position detection, which leads the monsters to the player. As is demonstrated in the code below, a loop will traverse every monster and find out whether the coordinate of the player is in certain range. If so, the walk method of the monster will be called to move the monster towards the player. (see Fig 2) Different types of monsters have different vision, which means different types could sense and follow the player within diverse range. And interestingly, the speed of each monster could fluctuate slightly within a certain range with the help from the random method, in order to make the monster clutch move naturally.

$$v_{\text{skeleton}} = \frac{1}{4}\sum_{i=1}^{4} \text{randint}(1{,}5)\ldots \qquad (1)$$

If not, all the monsters will eventually get overlapped together in one piece, for they all move in the same speed relatively.


```
for skeleton in skeletons.sprites():
    if not skeleton.dying:
        if player.x - 500 < skeleton.rect.x < player.x - 20:
            skeleton.walk('right')
        elif player.x + 10 < skeleton.rect.x < player.x + 500:
            skeleton.walk('left')
for executioner in executioners.sprites():
    if not executioner.dying:
        if player.x - 600 < executioner.rect.x < player.x - 20:
            executioner.walk('right')
        elif player.x + 10 < executioner.rect.x < player.x + 600:
            executioner.walk('left')
```
Fig 2. The detecting vision of several kinds of monsters.

Different backgrounds are set in the game. As for background switching, a variable named bgimage is used to keep record of the background image to be rendered and the game loop keeps detecting the x coordinate of the player every frame. When the player reaches the end of the screen, the background image will be switched and the x coordinate of the player will be set 0, during which the position of the monsters will be moved backwards relatively as well. (see Fig 3)

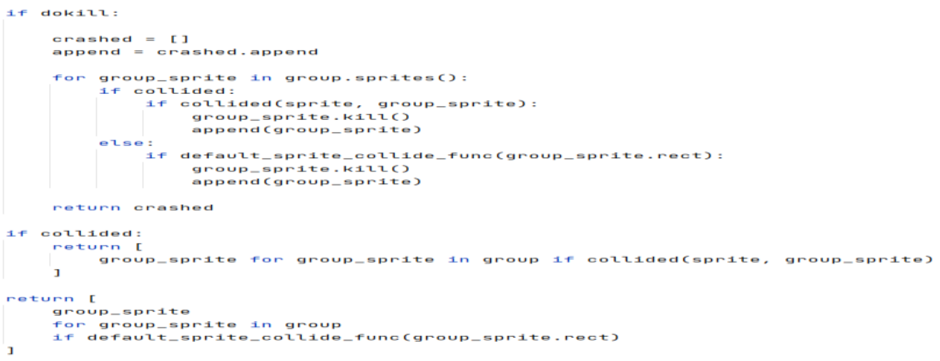


```python
if player.x >= WINDOW_WIDTH:
    bgimage_num += 1
    player.x = 0
    for skeleton in skeletons:
        skeleton.rect.x -= WINDOW_WIDTH
    for boss in bosses:
        boss.rect.x -= WINDOW_WIDTH
elif player.x <= 0 and bgimage_num >= 1:
    bgimage_num -= 1
    player.x = WINDOW_WIDTH
    for skeleton in skeletons:
        skeleton.rect.x += WINDOW_WIDTH
    for boss in bosses:
        boss.rect.x += WINDOW_WIDTH
if bgimage_num >= len(images):
    bgimage_num -= 1
    player.x = WINDOW_WIDTH
```

Fig 3. Background switching.

3 METHODOLOGY: COLLISION ALGORITHM

The authors adapted the built-in method for collision detection. As is shown below, the function will firstly find out whether a callback function defined by the programmer is passed to it. If so, the callback function will be called to find out whether the player collides with the monsters. Otherwise, the default collision detection function will be called to do so, which will detect collision via rect. (see Fig 4)



```python
if dokill:

    crashed = []
    append = crashed.append

    for group_sprite in group.sprites():
        if collided:
            if collided(sprite, group_sprite):
                group_sprite.kill()
                append(group_sprite)
        else:
            if default_sprite_collide_func(group_sprite.rect):
                group_sprite.kill()
                append(group_sprite)

    return crashed

if collided:
    return [
        group_sprite for group_sprite in group if collided(sprite, group_sprite)
    ]

return [
    group_sprite
    for group_sprite in group
    if default_sprite_collide_func(group_sprite.rect)
]
```

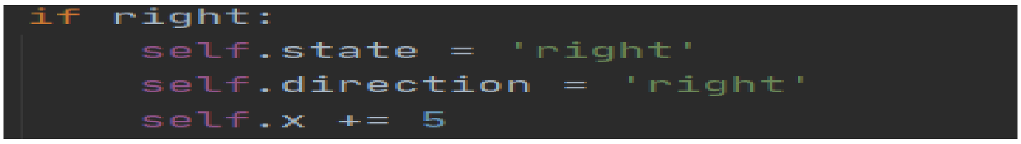
Fig 4. The collision algorithm based on Pygame.

4 METHODOLOGY: NUMERIC DESIGN IN GAME

For most games, numeric design is very important. This is not only related to whether the balance of the game is good, but also determines the experience of the player when playing this game. IMBA, which means “Imbalance”, is always a point to criticize for players of many games. Also, A good numeric design can also make the game more interesting. In the game, the authors use variety of ways to design values and other game mechanics to keep the game balanced and the experience good, including timing system based on the Pygame frames(or ticks), and a "dice" system based on summing multiple random numbers for a more balanced output.

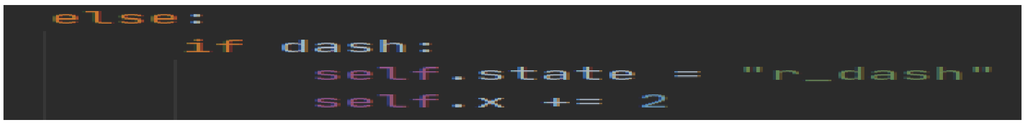
4.1 Timers: Dashing and its Cool-Down

“Roll” or “Flash” is often found in action games. Generally, this action supports the game character to move quickly (sometimes even just in a moment) in a particular direction for some distance. In most games, the character will not get attacked when performing this action. In the game, the authors successfully implemented “Dash” as the action like “Flash” in other games. Since the game hero have only 2 directions: left and right for the hero, the authors directly bind “Dash” with the normal running actions. The base speed of hero is 5px per tick (see Fig 5), and when player push left or right key with “Dash” (F key) together, the hero will get an extra speed of 2px per tick (see Fig 6). This extra value has changed many times during the programming. Both the authors and the test players have tested different extra speed and try to get a value that provides best experience.



```
if right:
    self.state = 'right'
    self.direction = 'right'
    self.x += 5
```

Fig 5. The base speed of running is 5px/tick.

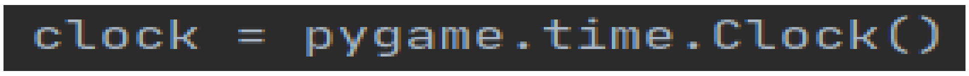


```
else:
    if dash:
        self.state = "r_dash"
        self.x += 2
```

Fig 6. “Dash” is related to both running action to determine the direction of extra speed.

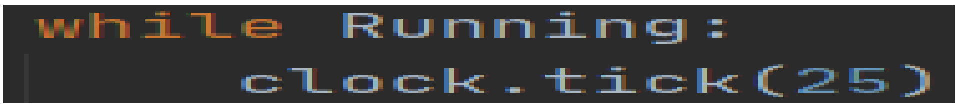
However, it’s imbalanced to allow unlimited dashing of player-controlled hero. So there are at least two restrictions that the authors need to implement: the largest dash time and the cool-down (CD in game terminology). So the authors choose pygame. time, which is a must for each Pygame projects. It provides several functions and utilities about the time of game, such as “get_tick” and “clock”. With these functions, the timers for dash itself and dash cool-down are successfully set up. (see Fig 7)

When a Pygame project begins, a pygame clock is needed, which determines the number of ticks (or frames) in a second. In the game, the authors make it 25 per sec. (see Fig 8)



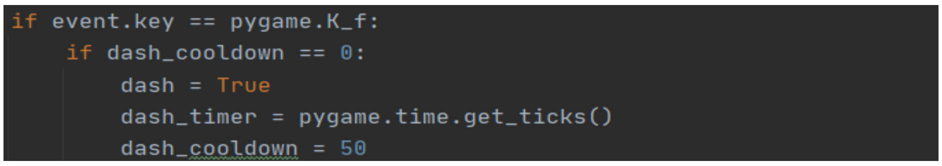
```
clock = pygame.time.Clock()
```

Fig 7. Pygame.time.Clock for game clock.



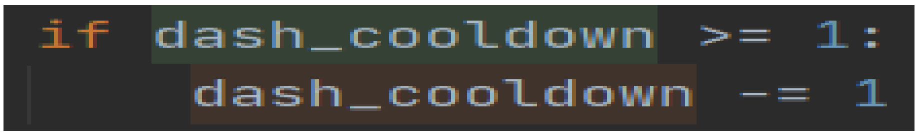
```
while Running:
    clock.tick(25)
```

Fig 8. Clock : 25 ticks per second.



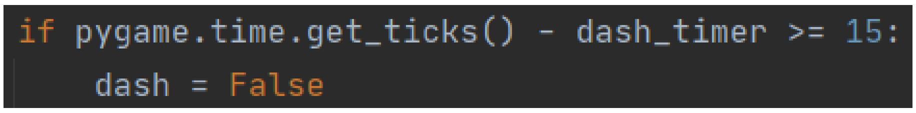
```
if event.key == pygame.K_f:
    if dash_cooldown == 0:
        dash = True
        dash_timer = pygame.time.get_ticks()
        dash_cooldown = 50
```

Fig 9. The cool-down timer 50 ticks, or 2 seconds.



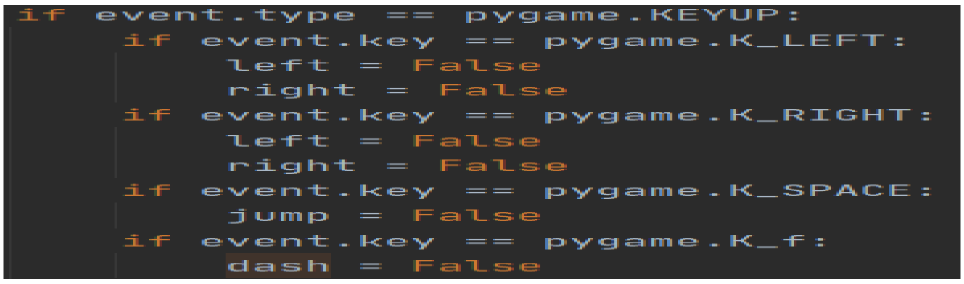
```
if dash_cooldown >= 1:
    dash_cooldown -= 1
```

Fig 10. Cool-down timer larger than 0 means CD is not finished.



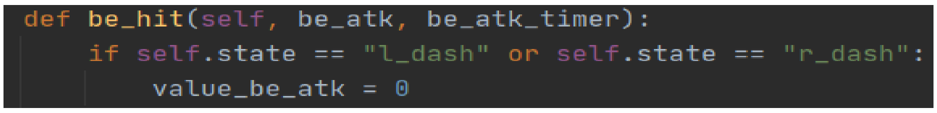
```
if pygame.time.get_ticks() - dash_timer >= 15:
    dash = False
```

Fig 11. In this and the previous pic, a timer(0.6s) for dash is set.



```
if event.type == pygame.KEYUP:
        if event.key == pygame.K_LEFT:
            left = False
            right = False
        if event.key == pygame.K_RIGHT:
            left = False
            right = False
        if event.key == pygame.K_SPACE:
            jump = False
        if event.key == pygame.K_f:
            dash = False
```

Fig 12. Also, dash can manually stopped.



```
def be_hit(self, be_atk, be_atk_timer):
    if self.state == "l_dash" or self.state == "r_dash":
        value_be_atk = 0
```

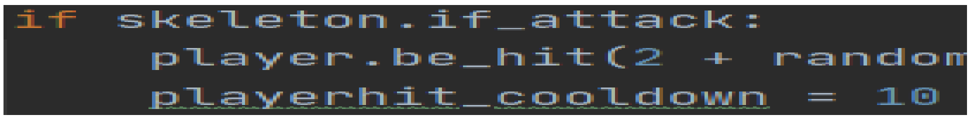
Fig 13. Dashing means invincible.

The implementation of two timers for dash is showed in Fig 9 and Fig 11. The authors have defined a timer for 15 ticks, or 0.6 sec, which limits the maximum duration of dash; That's to say, after 15 ticks, your dash will be forcibly disabled. Another timer is related to the cool-down of dash. During the cool-down time (50 ticks or 2 secs), you can not use dash to quickly get rid of enemies. Also, in the end of every running term (a tick) of game, if the cool-down timer is larger than 0, which means the cool-down is still going (see Fig 10), it will decrease at the rate of 1 per tick.

In particular, for players who prefer micro-control, they can also manually stop dashing before the dash timer ends(see Fig 12), which allows the hero dash for a shorter time and distance than normal actions. As mentioned at the beginning of this section, the "Flash" in the game generally comes with a short-time effect of invincibility. In this game, this feature is also implemented.(see Fig 13)

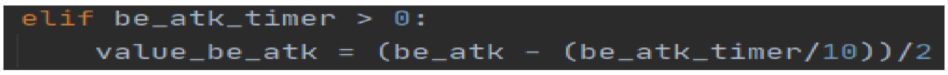
4.2 Shield frame after being attacked : another application of timer

In earlier versions of the game, some test players tried the demo and gave feedback about the hero losing life points at a too fast rate once he was approached by enemies. So the makers borrowed the "invincible frame" mechanic from some classic 2D platform adventure games and designed this special "shield frame". Instead of several ticks of invincibility just after being attacked for the hero, a timer-based damage reduction mechanism is applied to the game.



```
if skeleton.if_attack:
    player.be_hit(2 + random
    playerhit_cooldown = 10
```

Fig 14. Player be-hit timer,ranging from 0 to 10 tick(s).



```
elif be_atk_timer > 0:
    value_be_atk = (be_atk - (be_atk_timer/10))/2
```

Fig 15. Reduced damage when player be-hit timer larger than 0.

According to Fig 14, when the hero is attacked by any kind of enemies, the player be-hit timer will count down from 10 ticks. In these ticks or frames, the real value of damage taken by hero will be reduced. For example, the mathematical expectation of the attack of skeleton(a kind of monster in the game) equals to 2. If a skeleton continuously attack the hero in 2 ticks, the hero will take 2+(2-(10/10))/2 = 2.5 points of damage on health instead of 2*2=4(see Fig 15). That’s to say, 37.5% of damage were reduced to protect the hero who just be hit.

A more general arithmetic derivation of the above algorithm is as follows:

Let x be a the mathematical expectation of the damage of a certain kind of monster;

Then in the tick hero firstly get attacked and the next tick, the damage without “shield” equals to 2x;

In the 2nd being-attack tick, the actual damage with shield is

$$x + \frac{x - \frac{10}{10}}{2} = \frac{3}{2}x - \frac{1}{2} \dots \tag{2}$$

So the damage reducing ratio in these two ticks is

$$\frac{\frac{3}{2}x - \frac{1}{2}}{2x} = \frac{x+1}{4x} \dots \tag{3}$$

Since x ranges from 1 to INF, the damage reducing ratio ranges from 25% to 50%, increases with x.

What’s more, the timer will be reset to 10 every time hero gets attacked. So this shield with timer is useful to improve the game experience.

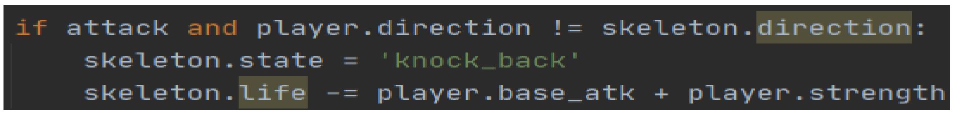
4.3 Strength: A more complex mechanism about “Attack”

In particular, for players who prefer micro-control, they can also manually stop dashing before the dash timer ends (see Fig 12), which allows the hero dash for a shorter time and distance than normal actions.

Another basic action of hero is “Attack”, which is necessary for executing enemies. This is also an important part of improving the game experience. There are too many IMBA happened among the values for different characters’ attack. The hero has a basic value of attacking damage, however according to the feedback from test players, it is boring that the attack damage just a constant.

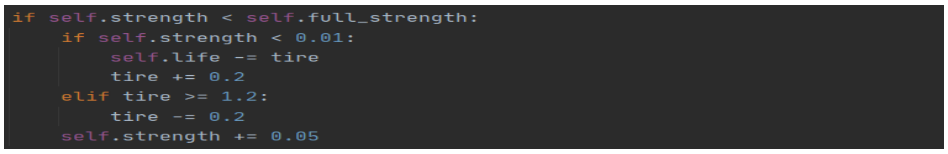
The makers also have considered to make the value a random number, however it turns out that players just roll dices to live or not. Now the actual attacking damage of hero is the sum of basic attack damage (actually too little to execute any enemy) and the strength value just when player attacks. According to the codes, the basic attack damage is 5, and the max strength hero can have is 15. That's to say, the hero can give out up to 4 times of basic damage. (see Fig 16)

Strength is all used each time player attacked and can restore automatically in a proper speed. When player's strength is too little for continuous time, player will become losing life, which is called "tiring".



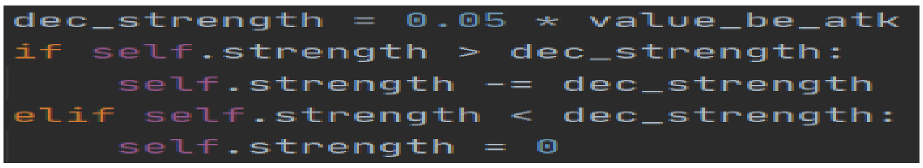
```
if attack and player.direction != skeleton.direction:
    skeleton.state = 'knock_back'
    skeleton.life -= player.base_atk + player.strength
```

Fig 16. The actual value of hero’s attack damage is the sum of base value and strength.



```
if self.strength < self.full_strength:
    if self.strength < 0.01:
        self.life -= tire
        tire += 0.2
    elif tire >= 1.2:
        tire -= 0.2
    self.strength += 0.05
```

Fig 17. The mechanism about strength.



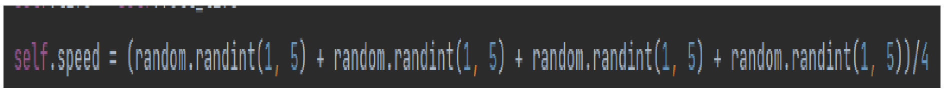
```
dec_strength = 0.05 * value_be_atk
if self.strength > dec_strength:
    self.strength -= dec_strength
elif self.strength < dec_strength:
    self.strength = 0
```

Fig 18. Strength damage.

As is shown in Fig 17, the longer hero run out of strength, the bigger life damage he will take. So it's not a good idea to keep pressing "K" key and try to attack continuously. Also, when player gets hurt, his strength will be decreased in a proportion. (see Fig 18)

4.4 Multiple random numbers: A more balanced dicing system

When a skeleton (a kind of monsters) is created, it will have a "speed" property, which defines its moving speed when running after player and try to attack him. Earlier versions use only one random integer generator to get a speed between 1 to 5. However, it is likely that 1 out of 5 skeletons run as fast as the hero(whose base speed is 5 too), which means the hero can hardly get rid of them. In order to get a more moderate speed for every skeletons, the generator is rewritten.



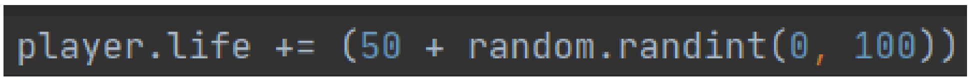
```
self.speed = (random.randint(1, 5) + random.randint(1, 5) + random.randint(1, 5) + random.randint(1, 5))/4
```

Fig 19. The actual value of hero’s attack damage is the sum of base value and strength.

In this new generator, it gets four rather than one (1,5) random integer in a time, and calculate the mean value of these four integers. According to the knowledge of probability theory, it's more likely to get a result just around 2.5 and there will be less skeletons who run too slow or too fast now. (see Fig 19)

4.5 Reward for executing enemies

In order to improve the game experience, a reward on health points is added into the game. (see Fig 20)



```
player.life += (50 + random.randint(0, 100))
```

Fig 20. The Executing rewards.

When hero successfully executed an enemy, he will get some health points restored. For a skeleton, 50 to 150 points; 80 to 180 for an executioner, and 120 to 220 for a boss. Also, every execution will give player several scores, which will be printed out when game is over.

5 EXPERIMENTS FOR THE GAME

All the three authors have tried this game. Also, Ding Ruochen Constantin, who is not smart on playing video games and other test players tested every versions of this game. Some screenshots in test playing are shown in Fig 21-23. According to the feedback from all the test players in the current version, it’s not a problem to pass Level 0; More than 3/4 plays successfully passed Level 1; And less than 1/2 plays passed Level 2.

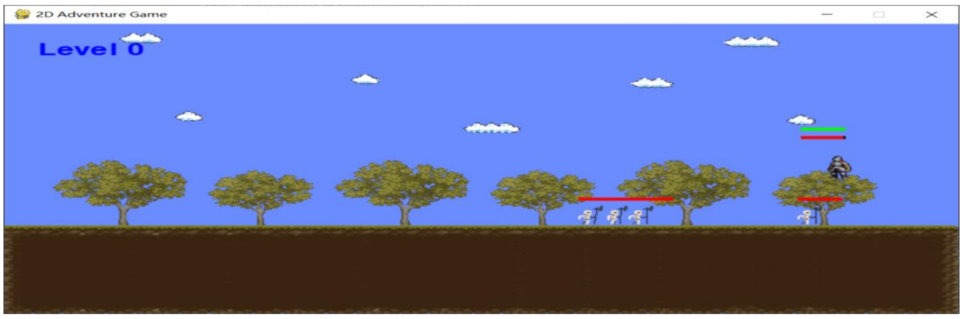


Fig 21. Test playing 1.

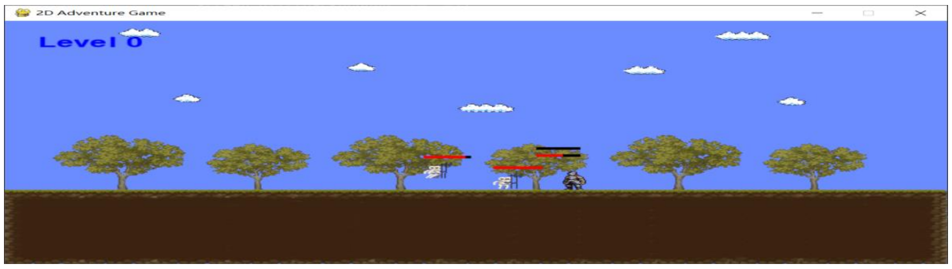


Fig 22. Test playing 2.

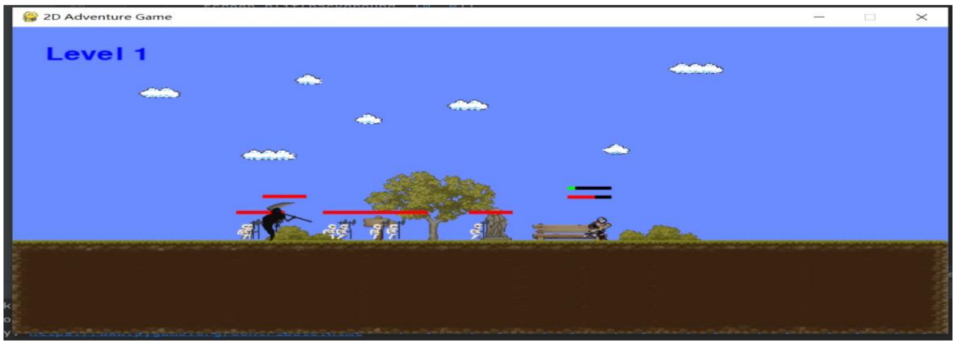


Fig 23. Test playing 3.

6 CONCLUSION

The authors successfully finished the implementation of the game prototype, mainly solving two problems: position and collision detecting, the cool-down timer for some game operations such as dash and attack. Also, some skills for game design in details of its mechanics. Many free assets [2-6] are applied to make the game demo more vivid and visual, including the hero [2], the monsters [3-5] and the background tile-set [6]. However, according to the definition of "platformer" game: having the player jumping from platform to platform, avoiding obstacles [7], the game demo is actually not strictly a Platformer. So another target for the authors is to implement the platforms in the game process and obstacles in the map. The authors will have the demo more completed in the future. In this article and game project, Pygame 2.1.2 [8,9] is applied as a Python package. Most of the image materials are quoted from [10], which is a website for independent game programmers.

Acknowledgement. Ding Ruochen, Su Yancheng and Xia Yifei contributed equally to this work and should be considered co-first authors.

REFERENCES

[1] Pygame. (2020) About Pygame - Pygame wiki. https://www.pygame.org/wiki/about

[2] Aamatniekss. Fantasy Knight (2021) Free pixelart animated character. https://aamatniekss.itch.io/fantasy-knight-free-pixelart-animated-character

[3] Jesse Munguia. (2020) Skeleton sprite pack. https://jesse-m.itch.io/skeleton-pack

[4] Kronovi-. (2021) Undead executioner. https://darkpixel-kronovi.itch.io/undead-executioner

[5] Clembod. Bringer of Death (Free). (2021) https://clembod.itch.io/bringer-of-death-free

[6] Cainos. Pixel art top down. (2021) https://cainos.itch.io/pixel-art-top-down-basic

[7] Game designing. Video games mechanics for beginners. (2022) https://www.gamedesigning.org/learn/basic-game-mechanics/

[8] René Dudfield et al. (2021) pygame 2.1.2.https://pypi.org/project/pygame/

[9] René Dudfield et al. (2021) pygame (the library) https://github.com/pygame/pygame

[10] Itch.io team. (2022) The game assets. https://itch.io/game-assets