MSc in Intelligent Robotics ELE00132M

C Programming

Exam Number: Y3915588

Design and Development of a Catch Ball Game

Table of Contents

1. Requirements

You are required to design and implement a computer game which is based on the simple game of catch. As the player, you need to guide an on-screen stick person to catch a ball thrown by a simulated (computer) thrower. The aim is to catch the ball before it hits the ground. The requirements in the report include the explicit part and the implicit part.

1.1 Explicit requirements

- the flight of the ball must be subject to the effect of gravity;
- the simulated (computer) thrower varies the speed, the angle and the rate at which the balls are thrown
- the game should keep a score of the number of successful catches over a set number of throws;
- the use of sound in the game;
- making use of both the mouse and keyboard to control the game;
- making good use of C features such as functions, structs and pointers;
- making good use of comments, formatted code and meaningful variable names but avoiding global variables.

1.2 Implicit requirements

- The game should be designed and developed in C programming language using CodeBlocks IDE
- The system should incorporate graphics, midi/amio libraries, and graphics libraries.
- The game should run on the Windows operating system.

To incorporate sound into your game, you can use the functionality provided by the amio lib. In particular, you could use a call-back design to supply real-time audio to your application (e.g. sound of the wind, the flight of the ball and the cheers of the crowd). Some good examples on how to achieve something like this can be found in the audio example files (audio01.c and other examples).

2. Analysis

2.1 Outline design - User entity

According to the requirements, what the user can do in the game includes:

- Catch ball
- Move character
- Control the game by clicking the mouse
- Control the game by pressing the keyboard
- Check current score

The main structure diagram is as follows:

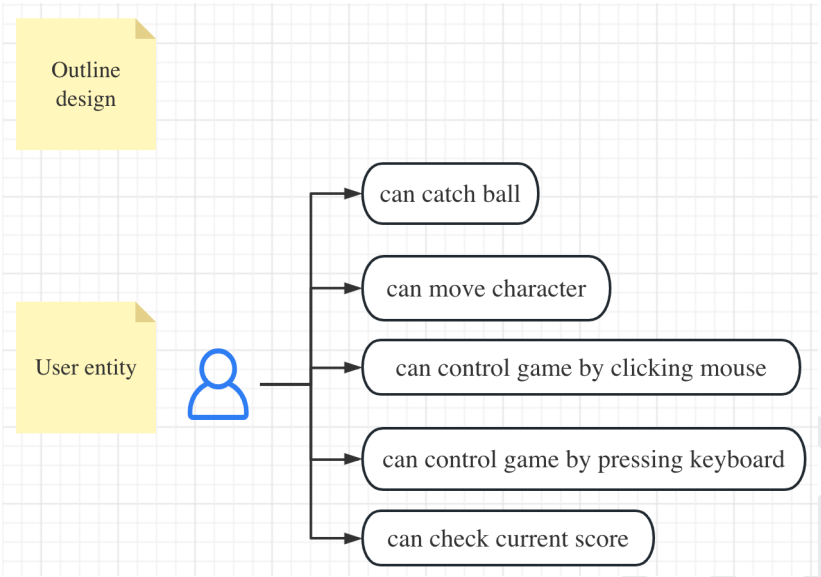


Figure 1 outline design - user entity

2.2 Outline design - Game entity

What the game itself can do in the game includes:

- Repaint canvas
- Show score
- Increase difficulty
- Tackle collision of ball and wall

The main structure diagram is as follows:

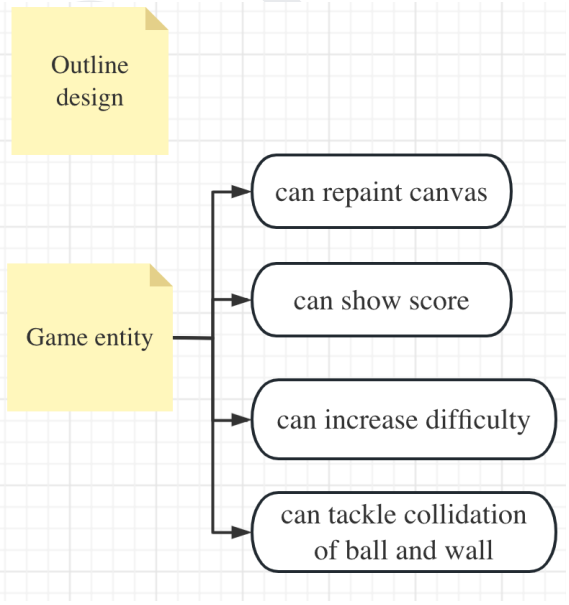


Figure 2 outline design - game entity

2.3 Outline design - User input

The user input includes keyboard and mouse click events.
The diagram is as follows:

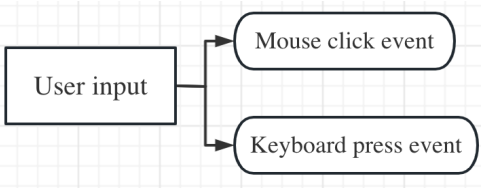


Figure 4 outline design - user input

2.4 Outline design - System output

The system output includes two main parts:

- Repaint canvas (current score, update ball and role position)
- Audio

The diagram is as figure 5 shows:

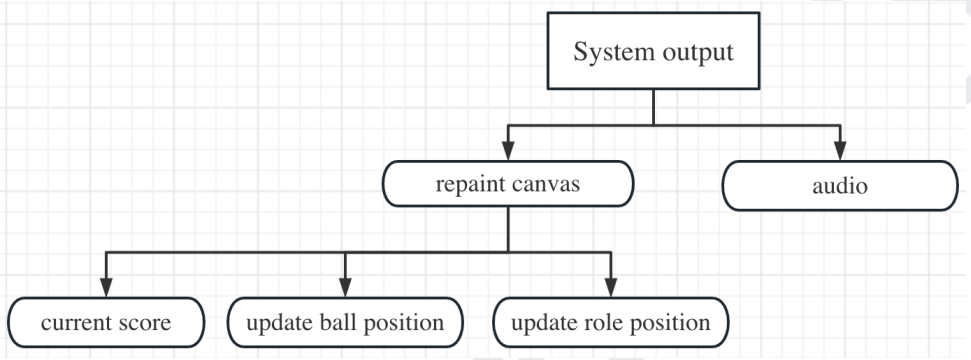


Figure 5 outline design - system output

2.5 Game panel design

I obeyed the outline design bullet-points and selects a theme of Santa Claus. The gift has 4 different pictures, and catching the 'Santa gift' would give 5 more marks.
The diagram is as follows:

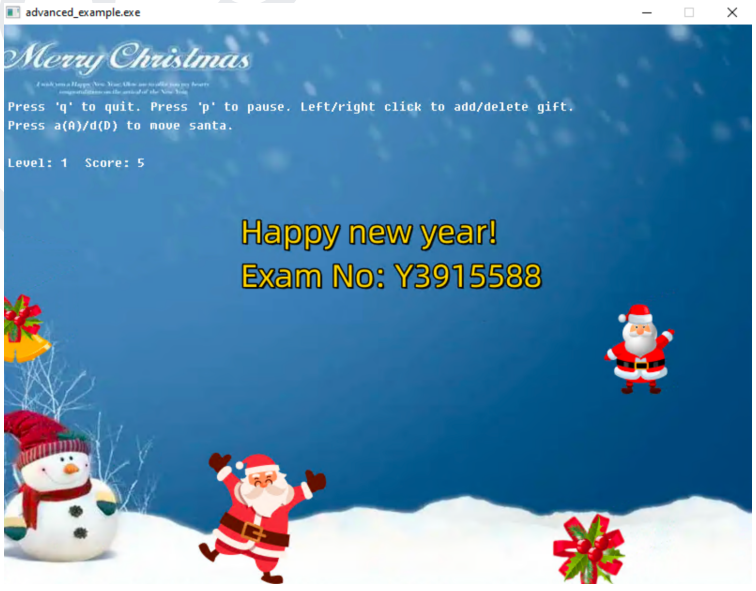


Figure 6 game panel design

2.6 mathematical formulas

$$P_x = P_{x0} + V_x t$$
$$P_y = P_{y0} - V_y t + (gt^2)/2$$

- (P_x, P_y) is the current position of the object;
- (P_{x0}, P_{y0}) was the initial position of the object;
- V_x was the velocity with which the object was thrown, in the x-direction in ms^{-1};
- V_y was the velocity with which the object was thrown, in the y-direction in ms^{-1};
- g is the gravitational pull ($9.81\mathrm{ms}^{-2}$);
- t is the time in seconds.

2.7 potential user

The user should be able to age over 8 and know basic control of the keyboard and mouse.

3. Specification - Analyse to design

3.1 Game feature

From the analysis above, I designed a Global struct to control the game panel, a Santa(Role) struct to control the motion of the role, a Ball struct to control the motion and some attributes of the ball, a AudioResourceBundle and PictureResourceBundle to manage the Audio and picture resources respectively.

The program must do:

- Throw the ball
- Show the score
- Have different difficulties
- Have a stick man can move and catch the ball

Optional:

- Play music
- Show a menu before the game start

The major features of this game, as shown in the below entity-relationship model (ER model).
The diagram is as follows:

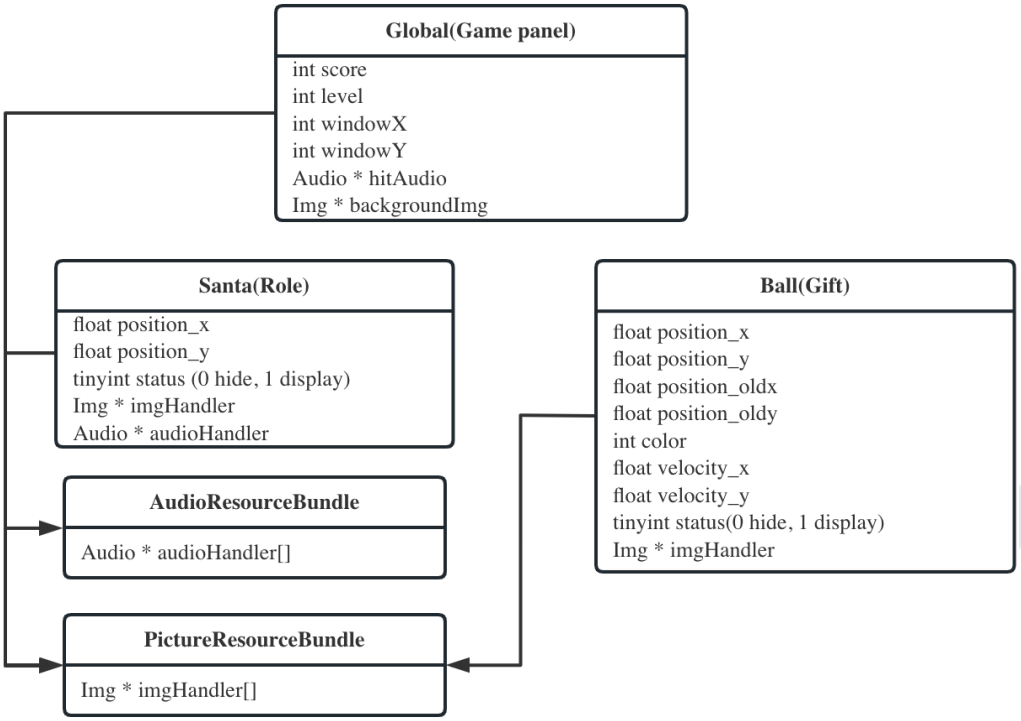


Figure 8 ER model

4. Design and implementation

4.1 User interface design

First, I designed a game panel, and then I designed a ball that comes from ‘ball generator’(a class to produce new balls) and a role which can move left or right. The ball must obey Newton's second law of motion.
Second, the player should control the role to catch the ball and gain marks.
Third, the canvas repaint event and keyboard event, which can move the role, must be asynchronous.
The diagram is as follows:

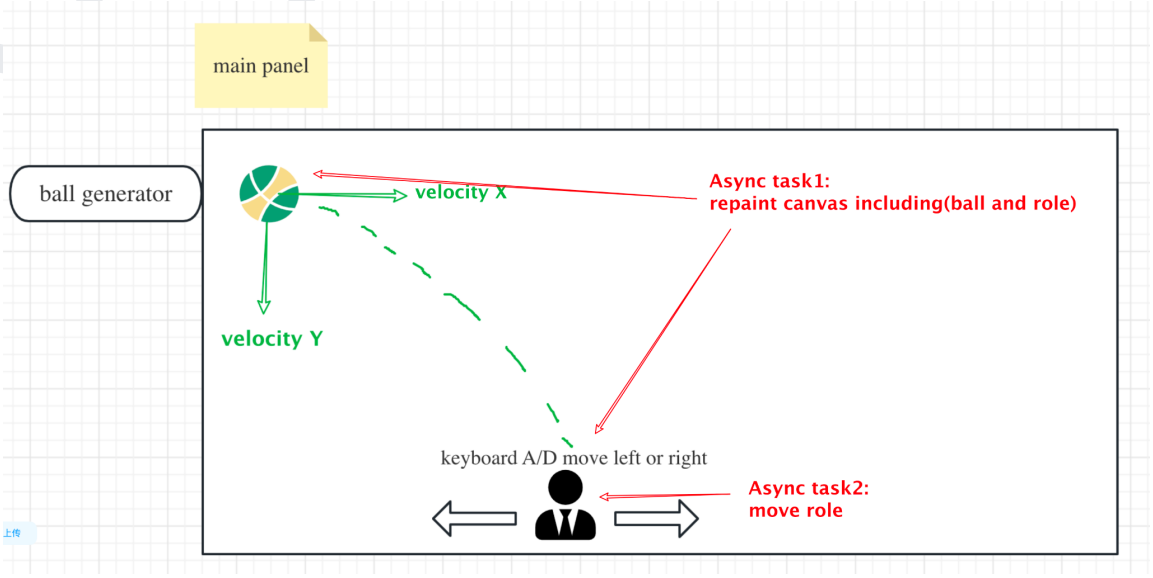


Figure 3 outline design - game design sketch

4.2 Project model structure - Domain-driven design

Because the project is simple, I only used one motion service(motion.c) to control the balls and role's motion and use the gamePanel service to update the panel display.
The diagram is as follows:

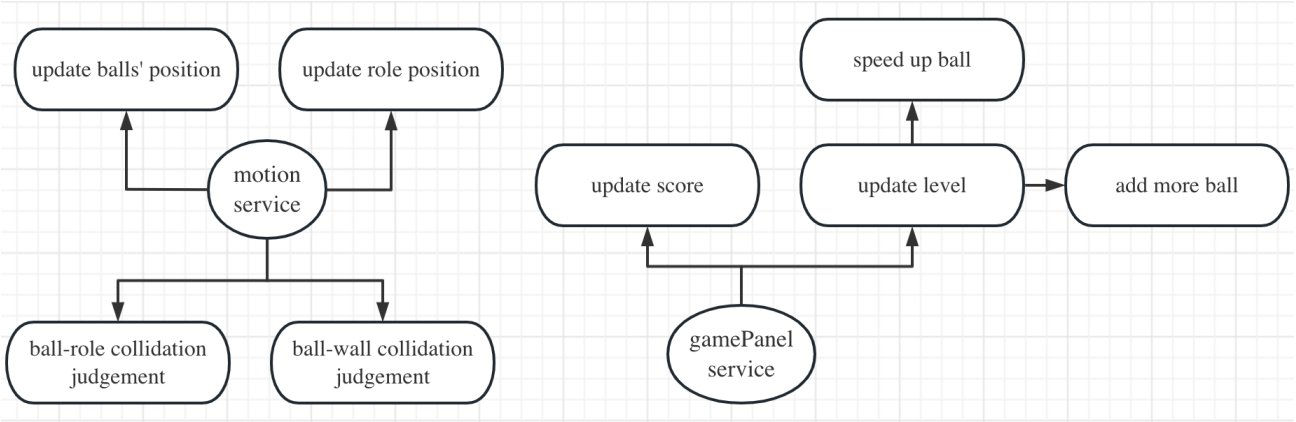


Figure 9 Domain-driven design

4.3 File structure

Because it's a game, I designed an assets folder to save audio and image resources. Then, I designed the service module to tackle complex logical calculations. Main.c is the entrance of the game. Global.h to save global variables and struct.h to save C structs.
The diagram is as follows:

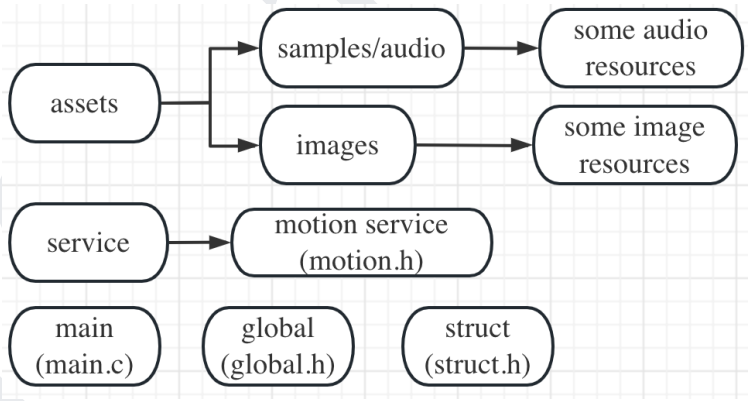


Figure 10 File structure

Functions used and their return values are accounted for in the table as follows:

Function Name	Description	Arguments	Return
main	Project entrance	void	int
initMenu	Init the games' menu	void	void
drawGift	Drawing the canvas frames with gifts' motion	Ball ball, BITMAP gifts[]	void

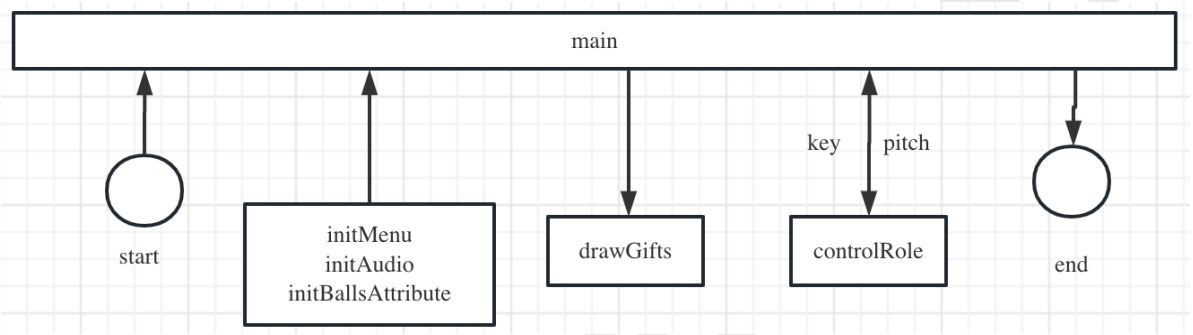
initBallsAttribute	Initing C struct balls' attributes including colours(use 0~4 to represent),coordinate-x,coordinate-y, coordinate-old-x,coordinate-old-y	Ball balls[]	void
initAudio	Pre-loading different audio resource, and set audio play loop	void	void
ball_update_new_pos	calculate new ball positions	Ball *ball	void
ball_update_old_pos	update old ball positions	Ball *ball	void
ball_hit_boundary	Check if the ball/gift have collided with the santa	Ball * ball, Role * role, int * number_of_balls, G * g	void
controlRole	use keyboard 'a','d' to control the motion of santa	Role *role	void

The major variables used by the above functions are listed in the below table.

Variable Name	Description	Type	Range
x_old	Last x coordinate	double	2.3e-308 ~ 1.7e+308
y_old	Last y coordinate	double	2.3e-308 ~ 1.7e+308
vx	X Velocity	double	2.3e-308 ~ 1.7e+308
vy	Y Velocity	double	2.3e-308 ~ 1.7e+308
color	Gifts' color or type	uint8	0 ~ 255
img	Img resource handler	BITMAP	
pos	Make data hold position	double	2.3e-308 ~ 1.7e+308

vel	Make data hold velocity	double	2.3e-308 ~ 1.7e+308
score	Player current score	int	-2,147,483,648 ~ 2,147,483,647
level	Game current level	int	-2,147,483,648 ~ 2,147,483,647
number_of_balls	Balls' count	int	-2,147,483,648 ~ 2,147,483,647

4.4 Logical flow



4.5 Major Algorithm design

4.5.1 Collide judgement

1. Get balls coordinate - ball (x, y)
2. Get Santa coordinate - santa/role (x, y)
3. If (SantaX - pictureRadius) < ballX < (SantaX + pictureRadius) and bally > (SantaY - pictureRadius), then collide

Pseudo-code:

```
Function collideDetect  (Use superposition of coordinates to judge collide)
if (ball->x_new > (role->x - 75) && ball->x_new < (role->x + 75) && ball->y_new > (role->y - 75)) Then
        printf("collide \n");
endif
```

4.5.2 Gain marks logic

1. Get ball object, ball(int color, double x, double y)
2. Judge the color, if color is equal to 3, gain 5 marks, otherwise gain 1 mark

Pseudo-code:

```
Function gainMarksAndPlaySound
if (ball->color == 3) then
        printf("yummy! \n");
        amio_add_sample_instance("laugh", PLAY_ONCE, 1.0);
        g->score += 5;
else
```

```
        g->score += 1;
endif
```

4.5.3 Level up logic

1. Get current score
2. If current score % 2 == 0, then level up
3. Add more balls or speed up the balls

Pseudo-code:

```
Function levelUp (get 2 more marks, level up(more difficult), speed up, add more balls)
if (g->score > 0 && (g->score % 2 == 0)) then
        printf("level up, speed up, oye! \n");
        amio_add_sample_instance("oye", PLAY_ONCE, 1.0);
        g->level += 1;
        ball->vy += 1;
        if ((*number_of_balls < MAX_NUMBER_BALLS))  then
                *number_of_balls += 1;
        endif
endif
```

The implementation part of the report contains any design changes made and their justification, and this part also contains all the source files and their contents.

4.6 Source Files

File name	File type	Contains(Functions/Prototype)
main.c	C source	Main function
global.h	C source	Global variables
struct.h	C source	All C struct
motion.h	C source	All motion functions

5. Verification and testing

5.1 White-box testing

I wrote some white-box testing functions to test collide and balls' motion and debug logs to help me fix bugs.

Python-whitebox-test-code:

```
void white_box_test_ball_hit_boundary(Ball * ball, Role * role, int * number_of_balls, G * g)
{
   if (ball->x_new > (role->x - 75) && ball->x_new < (role->x + 75) && ball->y_new > (role->y - 75)) //
Use superposition of coordinates to judge colide
   {
      printf("collide \n");
```

```
        amio_add_sample_instance("spectra", PLAY_ONCE, 1.0);
        ball->x_new = 0;
        ball->y_new = (double) rand_number(YMIN, YMAX) / 3;
        ball->vy = rand_number(1,3);
        printf("color: %d \n", ball->color);
        if (ball->color == 3) { // ball/gift type 3, get 5 more marks, and play special laugh audio
            printf("yummy! \n");
            amio_add_sample_instance("laugh", PLAY_ONCE, 1.0);
            g->score += 5;
        } else { // normally , a gift get 1 mark
            g->score += 1;
        }
        if (g->score > 0 && (g->score % 2 == 0)) { // get 2 more marks, level up(more difficult), speed up,
add more balls
            printf("level up, speed up, oye! \n");
            amio_add_sample_instance("oye", PLAY_ONCE, 1.0);
            g->level += 1;
            ball->vy += 1;
        }
    }
```

5.2 Black-box testing

5.2.1 Input and output - GUI

I programmed a keyboard event used to pause the game panel so that I could easily debug the procedure, including judge the next motion of gifts or the Santa role.
The diagram is as follows:

Figure 11 Black-box testing - Input and output - GUI

5.2.2 Input and output - SHELL

I also programmed a function called ‘log’ to help me print debug messages in the shell. Therefore, every time when the game becomes harder, I can easily get the current difficulty coefficients.
The diagram is as follows:

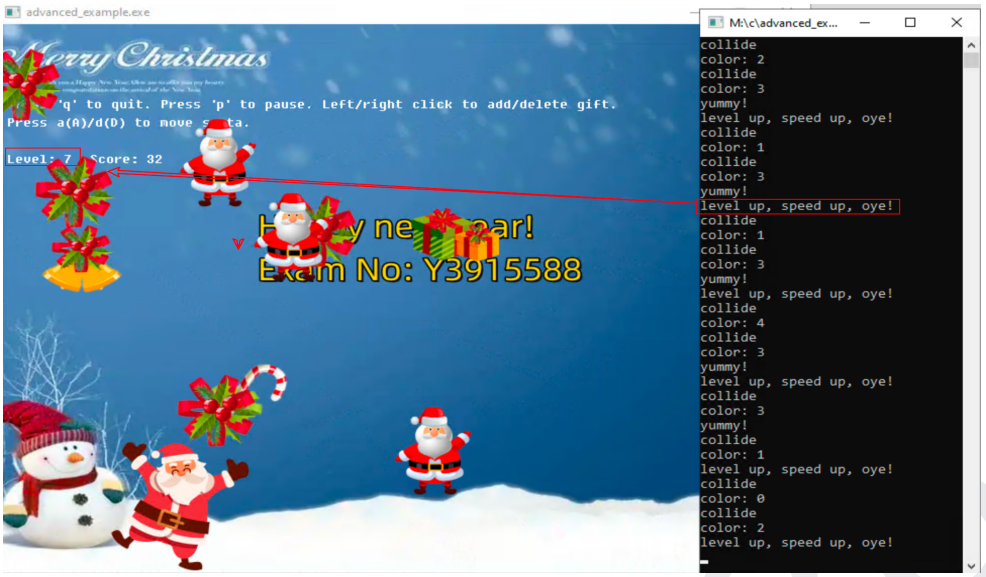


Figure 12 Black-box testing - Input and output - SHELL

5.3 Modifications - Design Changes

- At the beginning of the design, I was thinking use oop design model to apply to this project and use a santa class to control the role, but at last, I abandoned this plan and turned to a procedure-oriented model.
- I used to consider using c++ to program on some specific library because there are a lot of c++ open-source libraries. However, I turned to use C fully.

6. User Manual

6.1 Intended users

This game is suitable for people over the age of 8, and teens are the main customers of this game.

6.2 Installation Instructions

The code is already compiled to execute(.exe) files, so the player can directly run it.

6.3 System Requirements

Platform: Windows x32, x64
Memory: > 2GB
CPU: > 4

6.4 FAQ

How to gain difficulty? how to get scores?
1. Catching the gift will give you marks, and more marks will let the game becomes harder(gift drown speed up, throw multiple gifts at the same time).
2. Catching Santa Claus will give you 5 marks, and other gifts only give you 1 mark.