***The Gates Fighter***

**24780 Project Detailed Design**

**Instructor: Soji Yamakawa**

**Team 3**

**11/12/2017**

Product name: the gates fighter

Team members: Tianyu Li, Tianyi Shen, Wenyuan Zhang, Zhenglin Chen, Zhuohao Zhang

In this project, we will write a fighter game which is controlled by two players. The players will use the assigned keys to control the motion of their characters, which can be divided into a series functions such as moving, jumping and punching. So we make a Player class to convert all these functions into member functions. For each player, we need to store several properties, including position, direction, speed, HP, current state and so on. To prevent these parameters from being modified in the main function, we make these data members protected. Since we still need to get access to some of these parameters, we then create several accessors to get the value of the protected data member.

For the motion, we have three kinds of action.

1. Move. The player will command the character to move left or right by the keys. The motion will be updated every iteration in the main function.
2. Jump. The jump motion will be triggered once the player command the character to jump when the character is currently not jumping. The character will be subjected to the gravity so the vertical speed and position will change in every iteration. Then we need to check whether the character has been on the ground or not.
3. Punch. Similarly, we need to check whether the character is punching. The speed of the punch is a constant so the position of the fist will change in every iteration. Once the arm has been totally extended, the fist will then be drawn back. In that situation, we need to check whether the punch has been completed or not.

The main idea of the fighter game is to detect and evaluate whether the player has been hit and how much damage has been cost. In our case, we will measure the fist position of the player who has punched, let’s say Player 1. Then we need to know Player 2’s position so that we are able to determine whether Player 1 has hit Player 2 or not by evaluating Player 2’s control polygon. The damage to Player 2 will depend on the attack force of Player 1.

For the structure of this project, we will have 2 header files and 3 cpp files. The first two are the menu and definition of classes. Then we will have a main function to achieve all the functions. The following are the detailed explanations for each part.

The first one is the menu header. Here is the pseudo code for game-menu.h.

#include <stdio.h>

#include "fssimplewindow.h"

#include "ysglfontdata.h"

#include "game-menu.h"

/\*

This file describes what is contained in the menu works for this project

\*/

void GameMenu::RunGameMenu(void)

{

/\*

Show the initialization of the menu, take the key as input to decide whether

to go to the help page or directly start the game.

S means start the game -> player 1 choose character -> player 2 choose character

H means go to the help page

ESC means quit

\*/

}

void GameMenu::Help(void)

{

/\*

Illustrate the instruction of this game for both player 1 and player 2.

\*/

}

int GameMenu::Player1ChooseCharacter(void)

{

/\*

If start the game, player 1 first choose the character

return the character number that player 1 has chosen

\*/

}

int GameMenu::Player2ChooseCharacter(void)

{

/\*

Player 2 choose the character

return the character number that player 2 has chosen

\*/

}

Then we need to define the class that we are going to use. Here is the pseudo code for game-class.h.

#include <stdio.h>

#include <stdlib>

#include <time.h>

#include "fssimplewindow.h"

#include "game-class.h"

/\*

This file describe the design of our project, including

the player class, multiple body parts classes and their

member variables and functions

\*/

class Player

{

protected:

/\* 'x' and 'y' represent the x,y coordinate on the opengl window. \*/

int x,y;

/\* 'hp' indicates the state of health. \*/

int hp;

/\*

'attack' indicates that if the player is attacking others.

'jumpState' indicates that if the player is jumping in the air.

'punchState' indicates that if the player is punching.

'isHit' indicates that if the current player is hit by others.

\*/

bool attack, jumpState, punchState, IsHit;

/\*

'ay', 'vx' and 'vy' indicate the kinematics properties of the player.

'direction' indicates the direction that the player is facing.

\*/

int ay, vx, vy, direction,

/\*

'w1' and 'w2' are angular velocities of the arms.

'elbowAngle1' and 'elbowAngle2' and the current angle of the arms.

\*/

int w1,w2,elbowAngle1,elbowAngle2;

public:

/\* Default Construcor of the player. \*/

Player();

/\* The following functions are getters of protected member variables. \*/

const int getAttack();

const int getLeftBoundary();

const int getRightBoundary();

/\*

The following functions are modifiers based on different moves that the players can do:

1. move

2. jump

3. punch

4. being punched

\*/

/\*

change x

\*/

void Move(void);

/\* Following functions describe the action of jumping \*/

/\*

Initialize a jump by set jumpState = 1 and vy = 20.

\*/

void InitializeJumping(void);

/\*

check jumpState, return true if jumpState == 1;

\*/

bool IsJumping(void);

/\*

change y and vy

y += vy \* dt;

vy = ay \* dt;

\*/

void Jump(void);

/\*

Check if the player is on the ground.

Change vy = 0 if on the ground;

\*/

void CheckHitGround(void);

/\* Following functions describe the action of punching \*/

/\*

Initialize a punch by set punchState = 1.

\*/

void InitializePunching(void);

/\*

check punchState, return true if puchState == 1;

\*/

bool IsPunching(void);

/\*

change arm state w1, w2 ,elbowAngle1 and elbowAngle2

\*/

void Punch(void);

/\*

check if punch is finished, return ture if finished

\*/

void CheckFinishPunching(void);

/\* Following functions describe what happen when a player hit his/her opponent\*/

/\*

take the other player as input, calculate the distance and return true if

the punch could cause damage to this player.

\*/

bool IfPunchHit(const Player &opponent);

/\*

Change isHit to 1 if this player is punched by the other player.

\*/

void ChangeHitState();

/\*

Decrease or increse hp by 'amount'.

\*/

void HPchange(const int amount);

/\*

Drawing function of player, including arm.draw and leg.draw.

\*/

void Draw();

};

class Arm

{

// protected:

// int xa,ya,xh,yh // albow, hand

public:

/\*

Drawing function for arms.

\*/

void Draw(void);

};

class Leg

{

// protected:

// int x1,y1,x2,y2

public:

/\*

Drawing function for legs.

\*/

void Draw(void);

};

/\*

Game class, represents the whole game

\*/

class Game

{

public:

void Run(void);

};

/\*

Potential Background class

\*/

The last one is the pseudo code for the main function.

#include <game-class.h>

#include <game-menu.h>

#include <stdio.h>

#include <stdlib>

#include <time.h>

#include "fssimplewindow.h"

// pseudo code of the main program of gates fighter

int main(void){

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// initialize the background parameters

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// the first while loop prompts a menu in the window

// this menu allows the users to choose their characters

// after users made their choices, exit while loop

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// initialize the corresponding fighter objects based on user's choice

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// the second while loop is the main body of the game

/\*

check for user inputs, possible inputs are:

1. Esc exit the game

2. lateral movements (left or right).

In total there should be four cases, two for player 1 and two for player two

3. Jump In total, two cases.

4. Punch Each player can punch toward right and left, therefore, 4 cases.

\*/

/\*

if player1.IsPunching() == true

call Punch() to update arm position

if IfPunchHit()==true

change player2 states and properties

if CheckFinishPunching() == true

change player 1 punching state to 0/false

\*/

/\*

if player1.IsJumping() == 1

call Jump() to update player's position

if checkHitGround() == 1

change player's jumping state to 0

\*/

/\*

if player2.IsPunching() == true

call Punch() to update arm position

if IfPunchHit()==true

change player1 states and properties

if CheckFinishPunching() == true

change player 2 punching state to 0/false

\*/

/\*

if player2.IsJumping() == 1

call Jump() to update player's position

if checkHitGround() == 1

change player's jumping state to 0

\*/

/\*

Drawing everything by calling both player's Draw() function.

Draw() function will call corresponding body parts' draw functions.

\*/

// update game time

// check both players' health powers

// exit the second loop if one player's health power equals to 0 or time runs out

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// enters the third while loop which displays the result of the game

return 0;

}

In the main function, we also need to add a sound as part of the background. Here is the pseudo code for the sound in game-main.cpp.

#include "fssimplewindow.h"

#include "yssimplesound.h"

/\* include the yssimplesound.cpp \*/

in the main function

YsSoundPlayer player;

/\* make current and start after open a window \*/

player.MakeCurrent();

player.Start();

/\* add a folder reference \*/

FsChangeToProgramDir();

/\* load audio files \*/

/\* sound for punching \*/

if(YSOK!=propeller.LoadWav("punch.wav") &&

YSOK!=propeller.LoadWav("datafiles/punch.wav"))

{

printf("Error! Cannot load punch.wav!\n");

}

/\* sound for moving \*/

if(YSOK!=notice.LoadWav("move.wav") &&

YSOK!=notice.LoadWav("datafiles/move.wav"))

{

printf("Error! Cannot load notice.wav!\n");

}

/\* sound for jumping \*/

if(YSOK!=notice.LoadWav("jump.wav") &&

YSOK!=notice.LoadWav("datafiles/jump.wav"))

{

printf("Error! Cannot load notice.wav!\n");

}

/\* sound for bleeding \*/

if(YSOK!=notice.LoadWav("bleed.wav") &&

YSOK!=notice.LoadWav("datafiles/bleed.wav"))

{

printf("Error! Cannot load notice.wav!\n");

}

/\* prompt message \*/

printf("Keys:\n");

printf(" W up\_key..........Play a jump\n");

printf(" A S left right..............play move.\n");

printf(" SPACE ENTER..............play punch.\n");

printf("B............play background music");

printf("E............end background music");

/\* decide which sound to play, play once or continuously according to the situation\*/

while game is not over

if isPunching returns true:

play punch for once;

if ispuchned returns true:

play bleeding for once;

if B is pressed:

play backgroundmusic;

if E is pressed:

end background music;

if W or up\_key pressed:

play jump for once;

if A S left\_key right\_key is pressed:

play move until they are released;

/\* after the main loop(game over), stop the music \*/

stop background music;

/\* end of the program \*/

return;