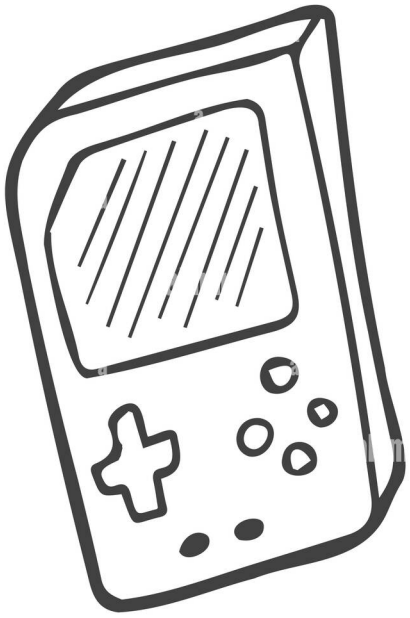
TABLE OF CONTENTS

PROBLEM STATEMENT

Design a C++ program for a gaming console that allows users to access and play multiple gaming programs. The program should provide a user-friendly interface that allows users to select their desired game from a list of available games. The console should be able to load and run the selected game. The program should also allow users to switch between different games without exiting the program. The gaming console should provide a seamless and immersive gaming experience to the user, with minimal lag or interruption.



MODULES OF THE PROJECT

CONSOLE:

A console program in C++ that accesses the programs of Snake, Hangman, and Flappy Bird can be created using the command line interface. The program starts by presenting the user with a menu of options to choose from, including playing Snake, Hangman, or Flappy Bird. When the user selects a game, the program launches the corresponding program using the system() function in C++, which allows the program to execute commands in the command line interface.

For example, to launch the Snake game program, the program would execute the command " system("g++ snake.cpp -o snake && snake")" in the command line interface. Similarly, to launch the Hangman game program, the program would execute the command " system ("g++ snake.cpp -o snake && snake")" , and to launch the Flappy Bird game program, the program would execute the command " system("g++ snake.cpp -o snake && snake")" . After the user finishes playing a game, the program returns to the menu of options, allowing the user to choose another game or exit the program. The console program can also include additional features, such as high scores or a leaderboard, that are shared between the games.

Overall, a console program in C++ that accesses the programs of Snake, Hangman, and Flappy Bird is a simple but effective way to create a game menu that allows the user to choose from multiple games. By using the system () function to execute commands in the command line interface, the program can launch other programs and integrate their functionality into a cohesive user experience.

SNAKE:

The snake game program in C++ is a classic game that is commonly used to teach the basics of programming. The game works by creating a window with a grid, where the snake is represented by a series of connected blocks. The objective of the game is to move the snake around the grid, eat the food, and avoid hitting the walls or its own body.

The snake game program in C++ starts by initializing the variables that are required for the game, such as the size of the window, the size of the grid, the speed of the game, and the initial position of the snake. Once the variables are initialized, the game enters into a loop that runs continuously until the game is over. Within the loop, the program checks for user input, such as pressing the arrow keys to move the snake in different directions.

The snake is moved by adding a new block at the front of the snake and deleting the block at the end of the snake. This gives the illusion that the snake is moving continuously. The program also checks for collisions between the snake and the walls, and between the snake and its own body. If a collision occurs, the game is over.

The food in the game is represented by a random block that appears on the grid. When the snake collides with the food block, the size of the snake is increased, and a new food block is generated at a random location on the grid. The score of the game is also incremented every time the snake eats a food block.

The snake game program in C++ also includes graphics that are displayed on the screen. The graphics are created using graphics libraries such as SDL or SFML. These libraries provide

functions for creating windows, drawing shapes, and displaying images. The graphics are updated every time the snake moves or eats a food block.
In addition to the basic game mechanics, the snake game program in C++ can be modified to add new features or improve the gameplay. For example, power-ups can be added to the game that provide temporary boosts to the snake's speed or size. Different levels can also be added to the game that increase the difficulty and challenge of the game.
Overall, the snake game program in C++ is a simple but engaging game that is a great way to learn the basics of programming. The game teaches concepts such as variables, loops, conditionals, and functions, and can be easily modified to suit different skill levels and interests. With its simple but addictive gameplay, the snake game program in C++ is sure to provide hours of entertainment and learning for both beginners and experienced programmers.

FLAPPY BIRD:

The Flappy Bird game program written in C++ is a simple game in which a bird jumps over a series of pipes. The game aims to make the bird jump by pressing the spacebar and reach as far as possible without colliding with pipes or the ground.

The Flappy Bird C++ game program begins by initializing the variables needed for the game, such as: B. Window size, bird size, pipe size and position, and game speed. Once the variables are initialized, the game enters a loop that runs continuously until the game ends. Inside the loop, the program checks for user input like this: B. Press Spacebar to make the bird jump. The bird moves while changing its position on the screen each time it runs a loop. When the user presses the spacebar, the bird repositions and jumps. The program also checks for collisions between birds and pipes and between birds and ground. If you hit it, it's game over.

In-game pipes are represented by rectangles that move across the screen from right to left. The program creates new pipes at random intervals, and the pipes are placed at random heights on the screen. There are also gaps between the pipes for birds to pass through. You can change the difficulty of the game by adjusting the size of the gap and the distance between the pipes.

The C++language game program Flappy Bird also includes on-screen graphics. Graphics are created in a graphics library such as SDL or SFML. These libraries provide functions for creating windows, drawing shapes, and displaying images. The graphics update each time the bird moves or a new whistle is generated.

In addition to the basic game mechanics, the Flappy Bird game program in C++ can be modified to add new functionality or improve gameplay. For example, you can add a power-up to your game that temporarily increases a bird's speed or jump height. You can also add different backgrounds and themes to your game to make it visually appealing. Overall, the Flappy Bird C++ game program is a fun and challenging game that can be easily created by beginners learning the basics of programming. Games teach concepts like variables, loops, conditions, and functions, and are easily adaptable to different skill levels and interests. With simple yet addictive gameplay and colorful graphics, his Flappy Bird game program in C++ language will provide hours of entertainment and learning for both beginners and experienced programmers.

HANGMAN:

The Hangman game program in C++ language is a popular word game where you guess the hidden word letter by letter. The game is played by randomly choosing a word and showing the player how many letters the word contains. The player then guesses the letter and if the letter is correct it appears in the correct place in the word. If the letters are incorrect, the body parts of the stickman will be drawn, starting with the head, followed by the body, arms, and legs. The object of the game is to guess the word before the stickman is completely drawn. A Hangman game program in C ++starts by initializing the variables needed for the game: B. Word list, number of guesses allowed, stick figure body parts. Once the variables are initialized, the program randomly selects a word from the word list and displays to the player how many letters the word contains. Then, when the player types a letter to guess, the program checks if that letter is in the word.

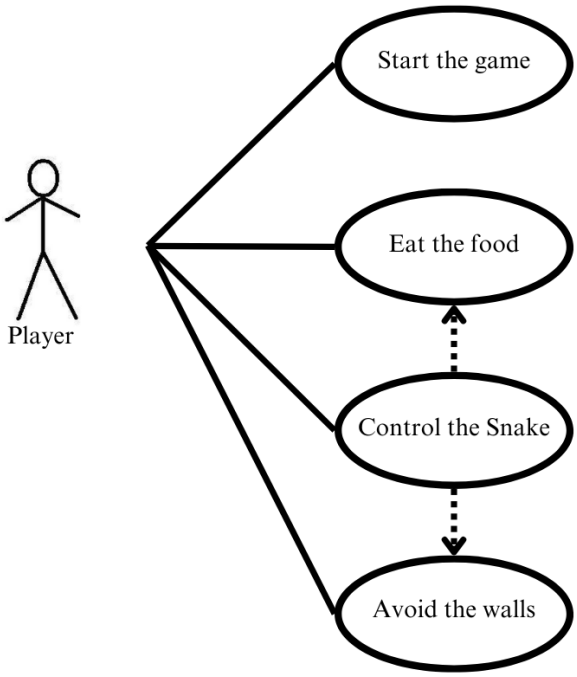
If the letter is in a word, the program will display the letter in the correct place in the word. If the letters don't appear in the word, the program draws the body parts of the stickman, starting with the head. The program also tracks the number of guesses left by the player and displays a message when the player has exhausted all guesses.

The C++ language Hangman game program also includes on-screen graphics. Graphics are created in a graphics library such as SDL or SFML. These libraries provide functions for creating windows, drawing shapes, and displaying images. The graphics update each time the player makes a guess or draws a stickman body part. In addition to the basic game mechanics, the C++ language Hangman game program can be modified to add new features or improve gameplay. For example, a program might include word categories such as animals, countries, or sports. The program may also include hints that provide clues to hidden words and a scoring system that rewards players for guessing words with fewer guesses.

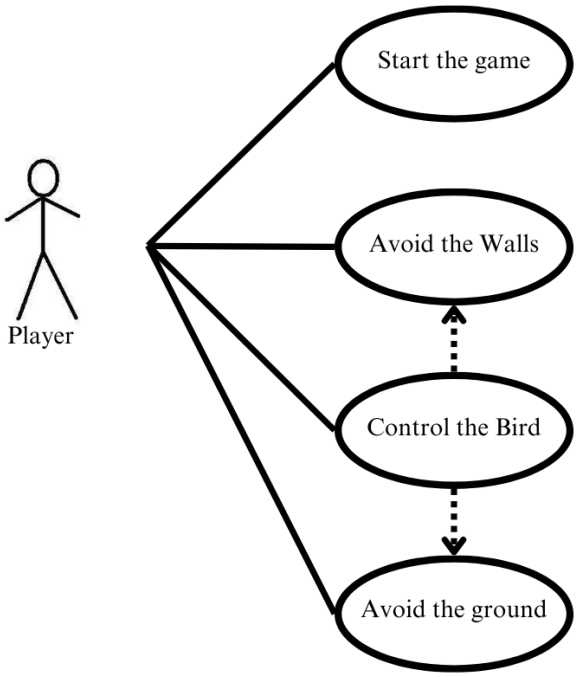
Overall, the Hangman game program in C++ is a fun and educational game that can be easily created by beginners learning the basics of programming. Games teach concepts like variables, loops, conditions, and functions, and are easily adaptable to different skill levels and interests. With simple yet engaging gameplay and customizable features, the C++ language Hangman game program provides hours of entertainment and learning for both novice and experienced programmers.

DIAGRAMS
USE CASE DIAGRAMS

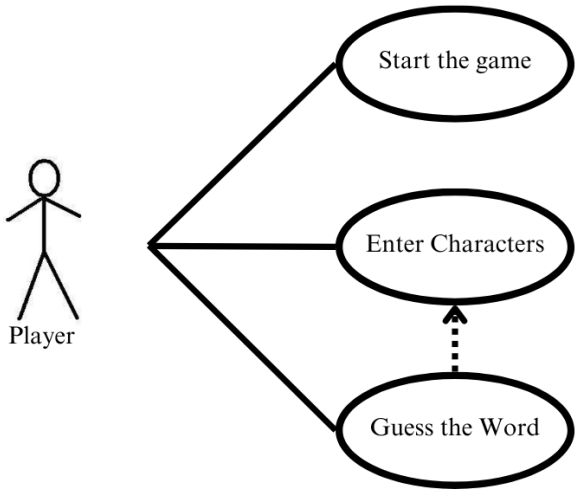
Snake



Flappy Bird

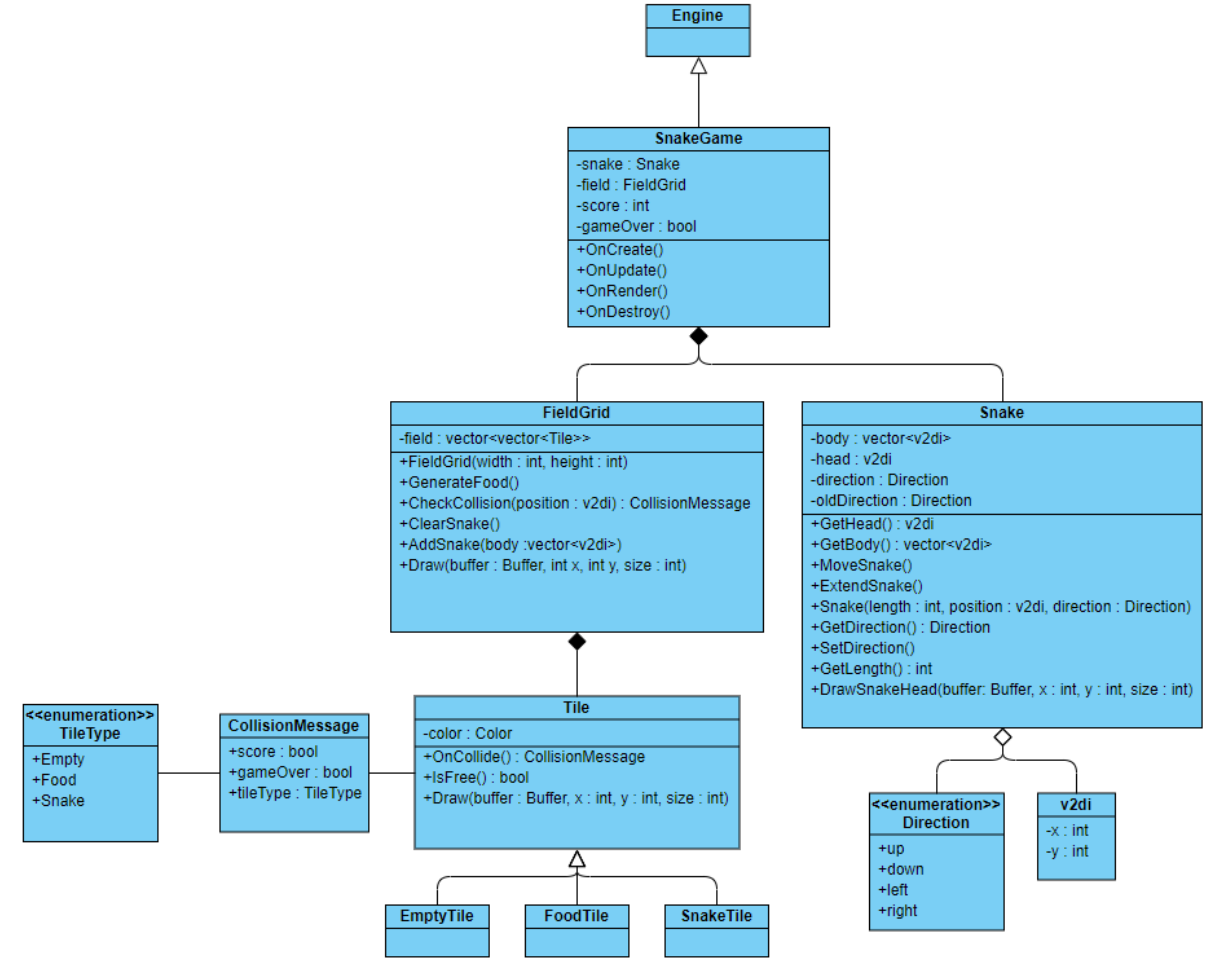

Start the game
Avoid the Walls
Control the Bird
Avoid the ground
Player

Hangman

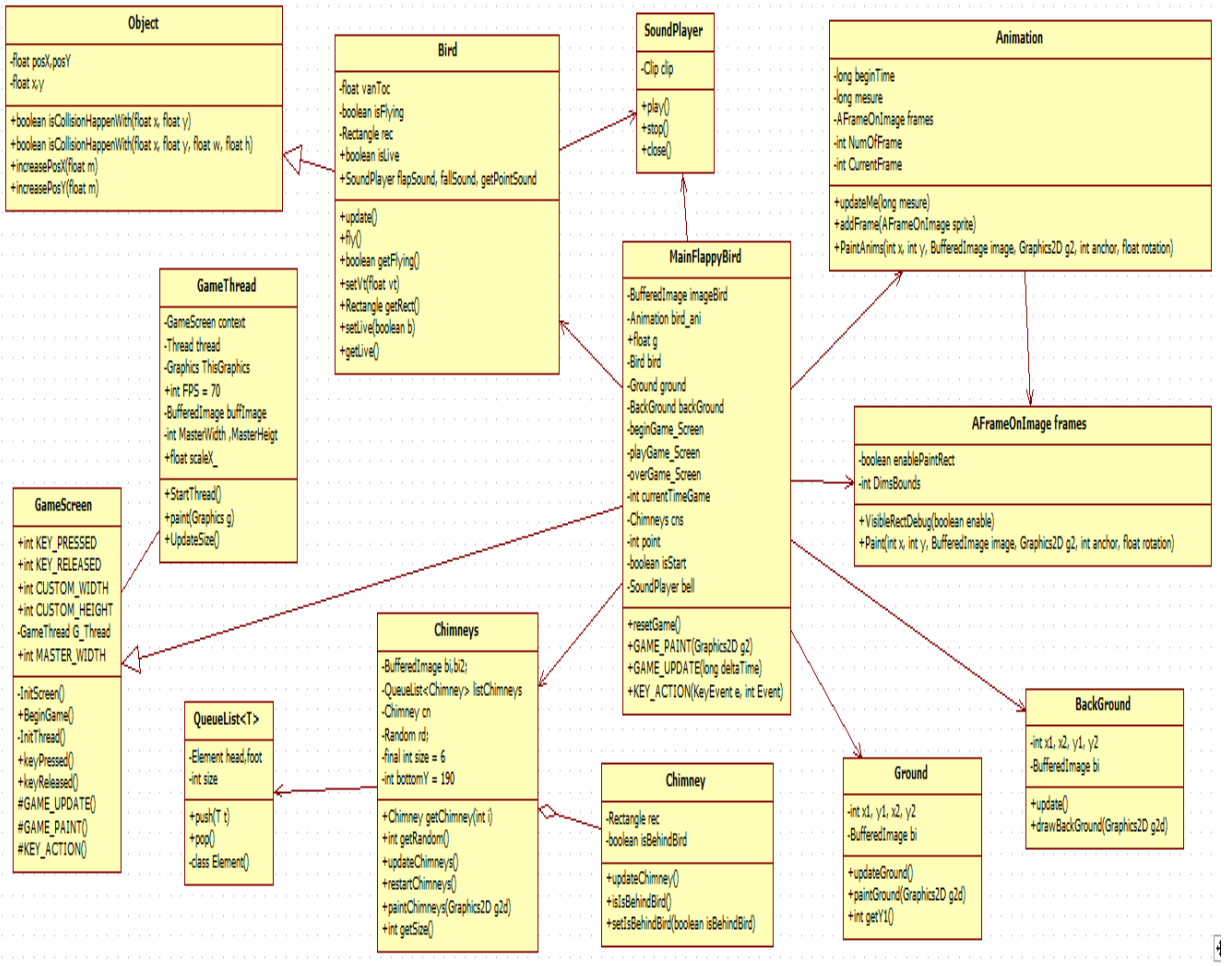


CLASS DIAGRAMS

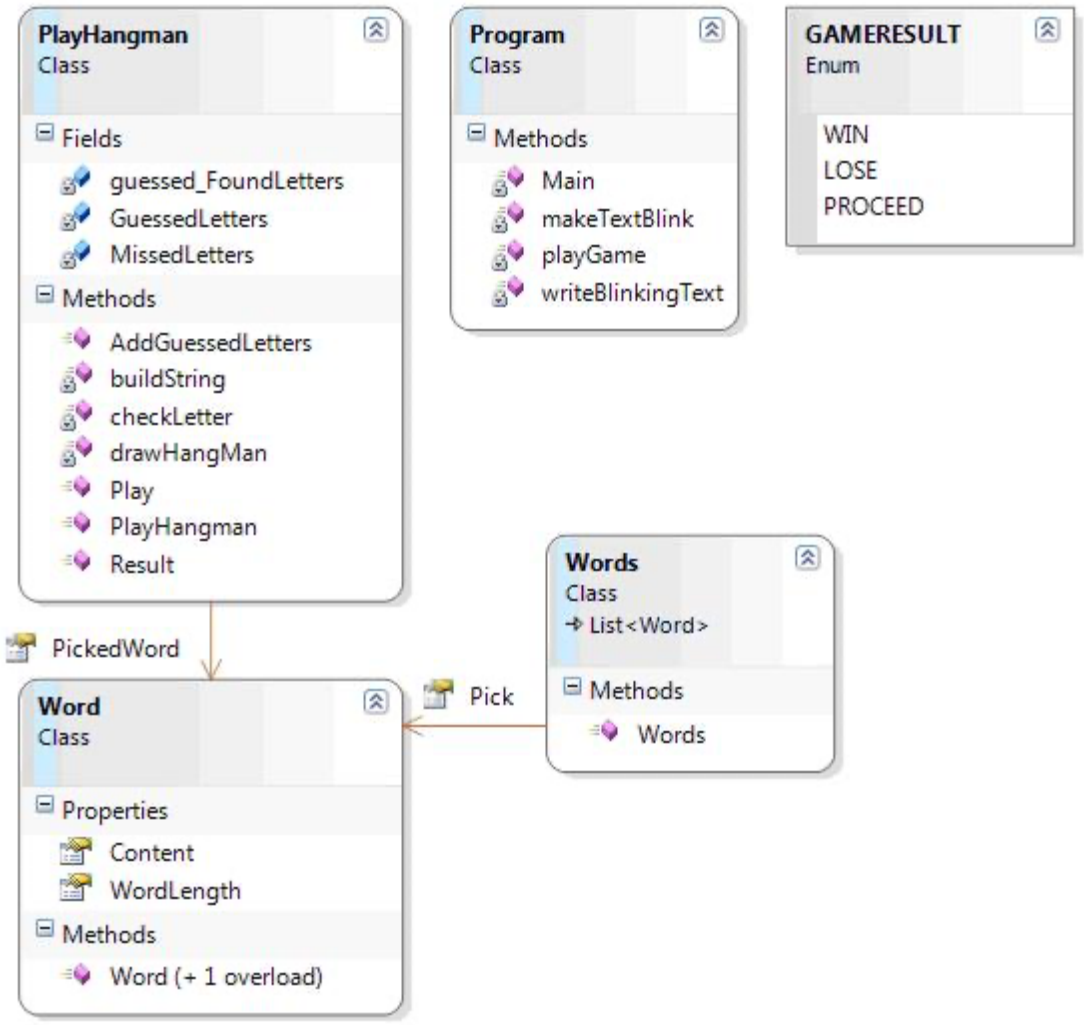
Snake



Flappy Bird

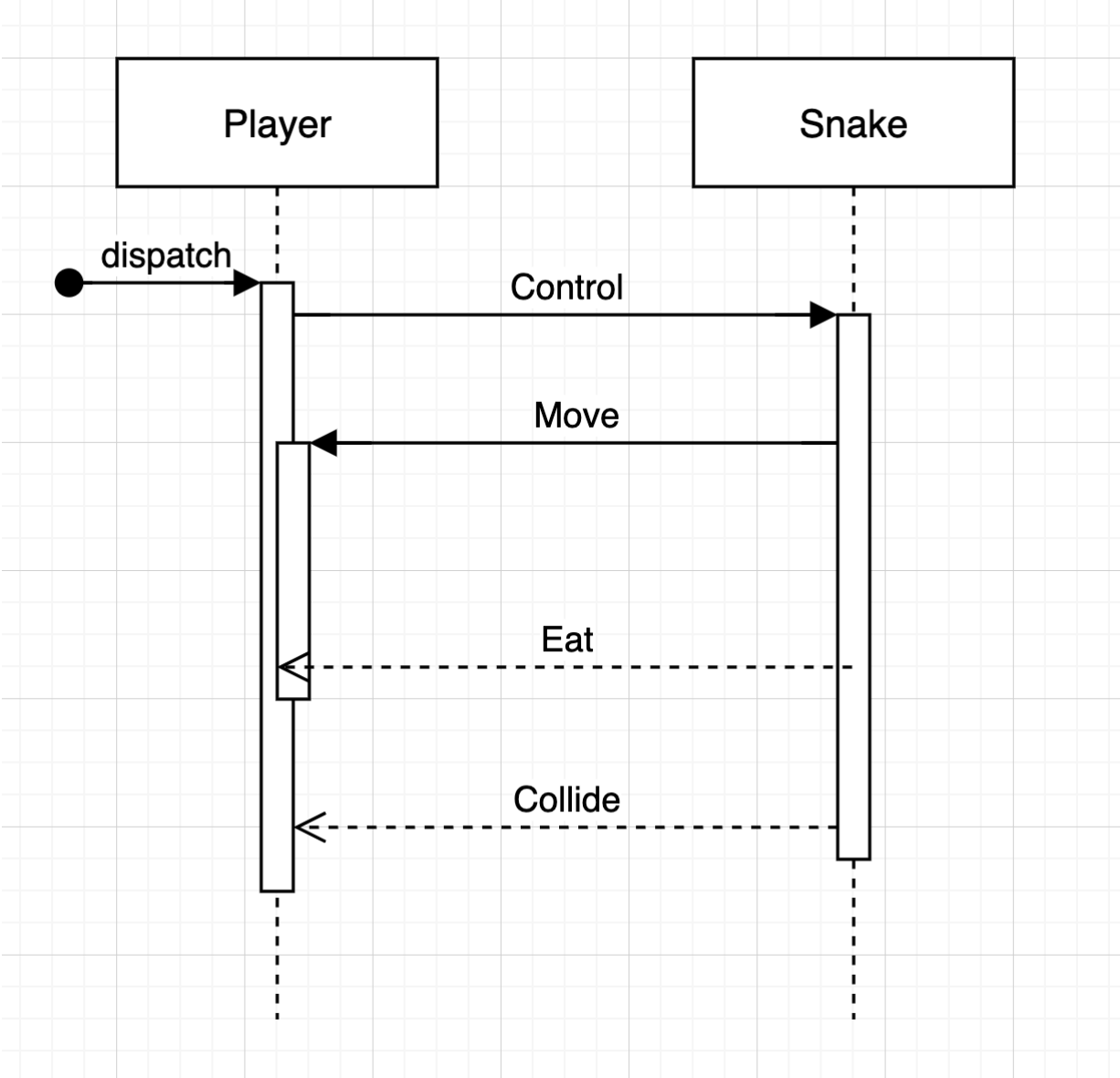

Object
-float posX,posY
-float x,y
+boolean isCollisionHappenWith(float x, float y)
+boolean isCollisionHappenWith(float x, float y, float w, float h)
+increasePosX(float m)
+increasePosY(float m)
Bird
-float vanToc
-boolean isFlying
-Rectangle rec
+boolean isLive
+SoundPlayer flapSound, fallSound, getPointSound
+update()
+fly()
+boolean getFlying()
+setVt(float vt)
+Rectangle getRect()
+setLive(boolean b)
+getLive()
SoundPlayer
-Clip clip
+play()
+stop()
+close()
Animation
-long beginTime
-long mesure
-AFrameOnImage frames
-int NumOfFrame
-int CurrentFrame
+updateMe(long mesure)
+addFrame(AFrameOnImage sprite)
+PaintAnims(int x, int y, BufferedImage image, Graphics2D g2, int anchor, float rotation)
GameThread
-GameScreen context
-Thread thread
-Graphics ThisGraphics
+int FPS = 70
-BufferedImage buffImage
-int MasterWidth ,MasterHeigt
+float scaleX_
+StartThread()
+paint(Graphics g)
+UpdateSize()
MainFlappyBird
-BufferedImage imageBird
-Animation bird_ani
+float g
-Bird bird
-Ground ground
-BackGround backGround
-beginGame_Screen
-playGame_Screen
-overGame_Screen
-int currentTimeGame
-Chimneys cns
-int point
-boolean isStart
-SoundPlayer bell
+resetGame()
+GAME_PAINT(Graphics2D g2)
+GAME_UPDATE(long deltaTime)
+KEY_ACTION(KeyEvent e, int Event)
AFrameOnImage frames
-boolean enablePaintRect
-int DimsBounds
+VisibleRectDebug(boolean enable)
+Paint(int x, int y, BufferedImage image, Graphics2D g2, int anchor, float rotation)
GameScreen
+int KEY_PRESSED
+int KEY_RELEASED
+int CUSTOM_WIDTH
+int CUSTOM_HEIGHT
-GameThread G_Thread
+int MASTER_WIDTH
-InitScreen()
+BeginGame()
-InitThread()
+keyPressed()
+keyReleased()
#GAME_UPDATE()
#GAME_PAINT()
#KEY_ACTION()
Chimneys
-BufferedImage bi,bi2;
-QueueList<Chimney> listChimneys
-Chimney cn
-Random rd;
-final int size = 6
-int bottomY = 190
+Chimney getChimney(int i)
+int getRandom()
+updateChimneys()
+restartChimneys()
+paintChimneys(Graphics2D g2d)
+int getSize()
QueueList<T>
-Element head,foot
-int size
+push(T t)
+pop()
-class Element()
Chimney
-Rectangle rec
-boolean isBehindBird
+updateChimney()
+isIsBehindBird()
+setIsBehindBird(boolean isBehindBird)
Ground
-int x1, y1, x2, y2
-BufferedImage bi
+updateGround()
+paintGround(Graphics2D g2d)
+int getY1()
BackGround
-int x1, x2, y1, y2
-BufferedImage bi
+update()
+drawBackGround(Graphics2D g2d)

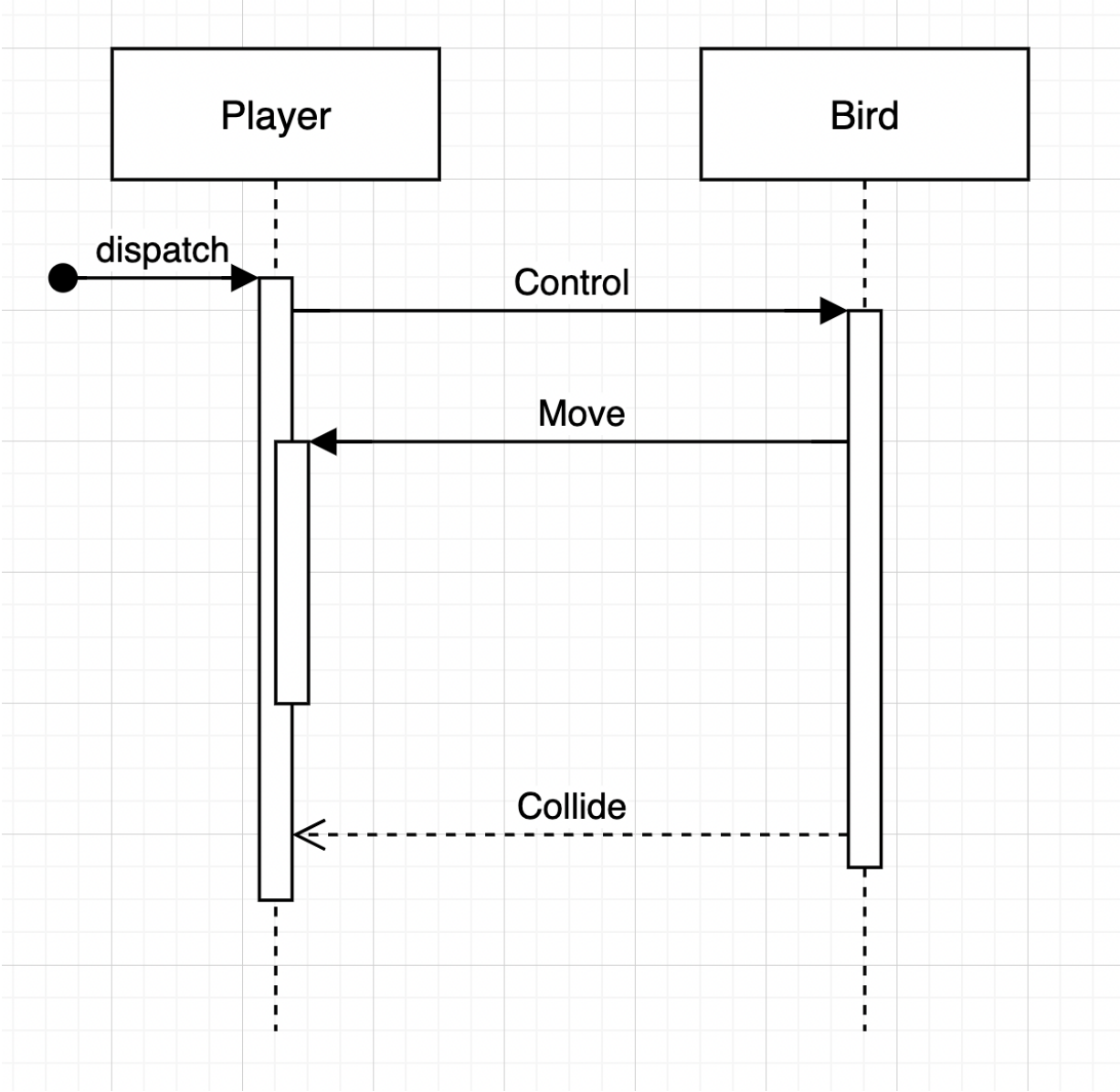
Hangman

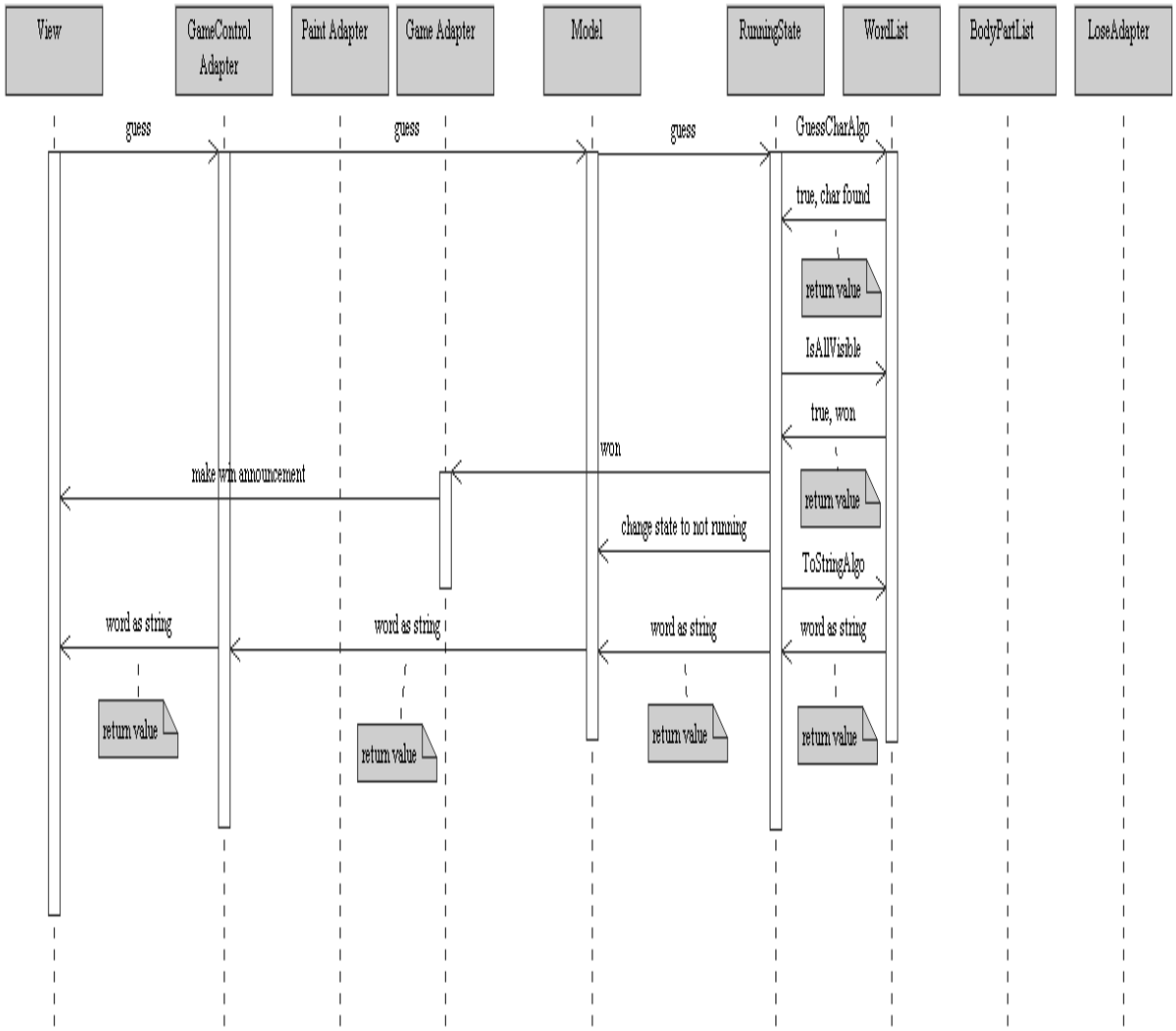

PlayHangman
Class
Fields
guessed_FoundLetters
GuessedLetters
MissedLetters
Methods
AddGuessedLetters
buildString
checkLetter
drawHangMan
Play
PlayHangman
Result
Program
Class
Methods
Main
makeTextBlink
playGame
writeBlinkingText
GAMERESULT
Enum
WIN
LOSE
PROCEED
Words
Class
List<Word>
Methods
Words
PickedWord
Pick
Word
Class
Properties
Content
WordLength
Methods
Word (+ 1 overload)

SEQUENCE DIAGRAMS

Snake

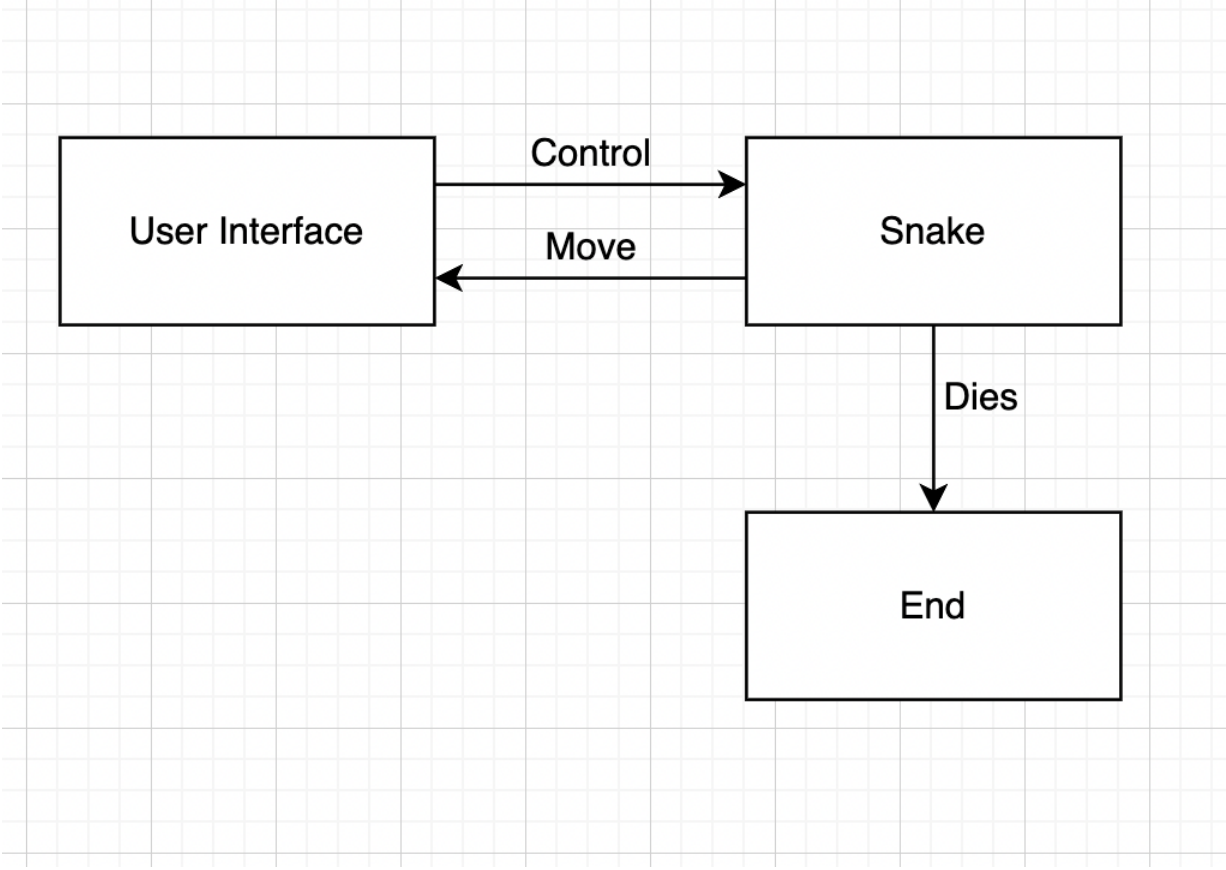



Player
Bird
dispatch
Control
Move
Collide

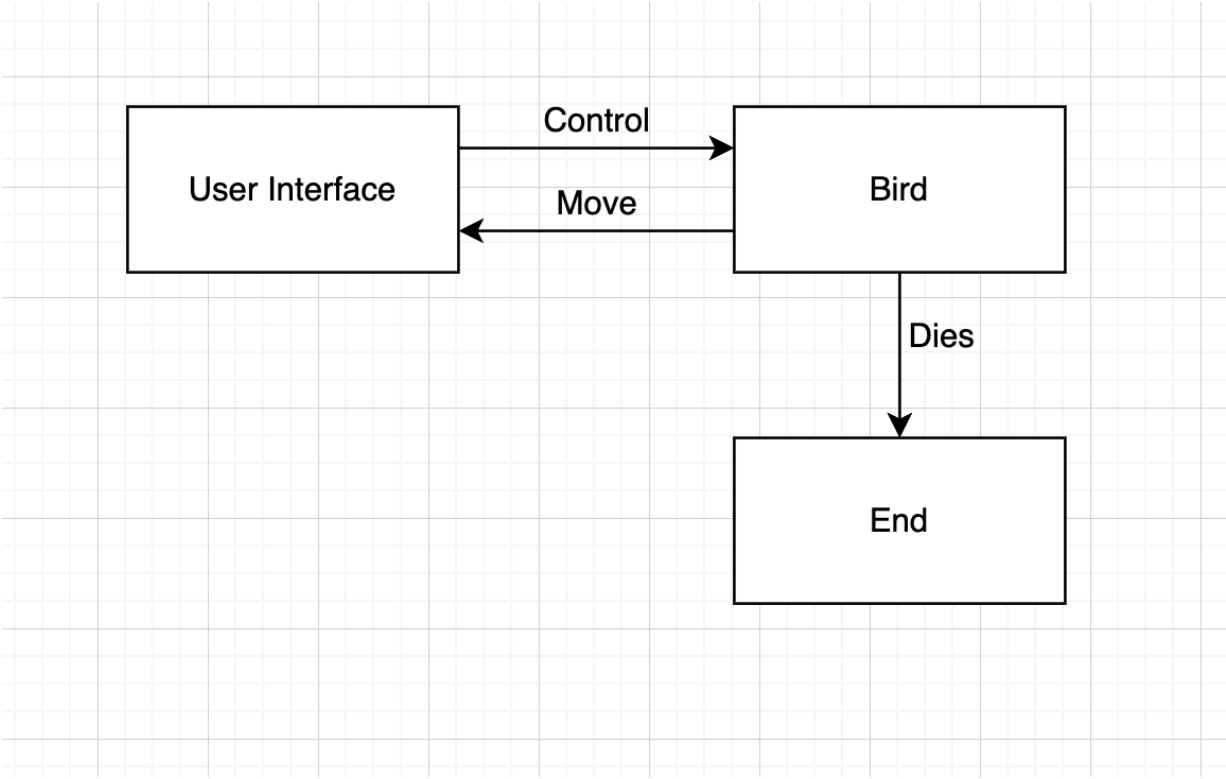

View
GameControl Adapter
Paint Adapter
Game Adapter
Model
RunningState
WordList
BodyPartList
LoseAdapter
guess
guess
guess
GuessCharAlgo
true, char found
return value
IsAllVisible
true, won
return value
won
make win announcement
change state to not running
ToStringAlgo
word as string
word as string
word as string
word as string
return value
return value
return value
return value

COLLABORATION DIAGRAMS

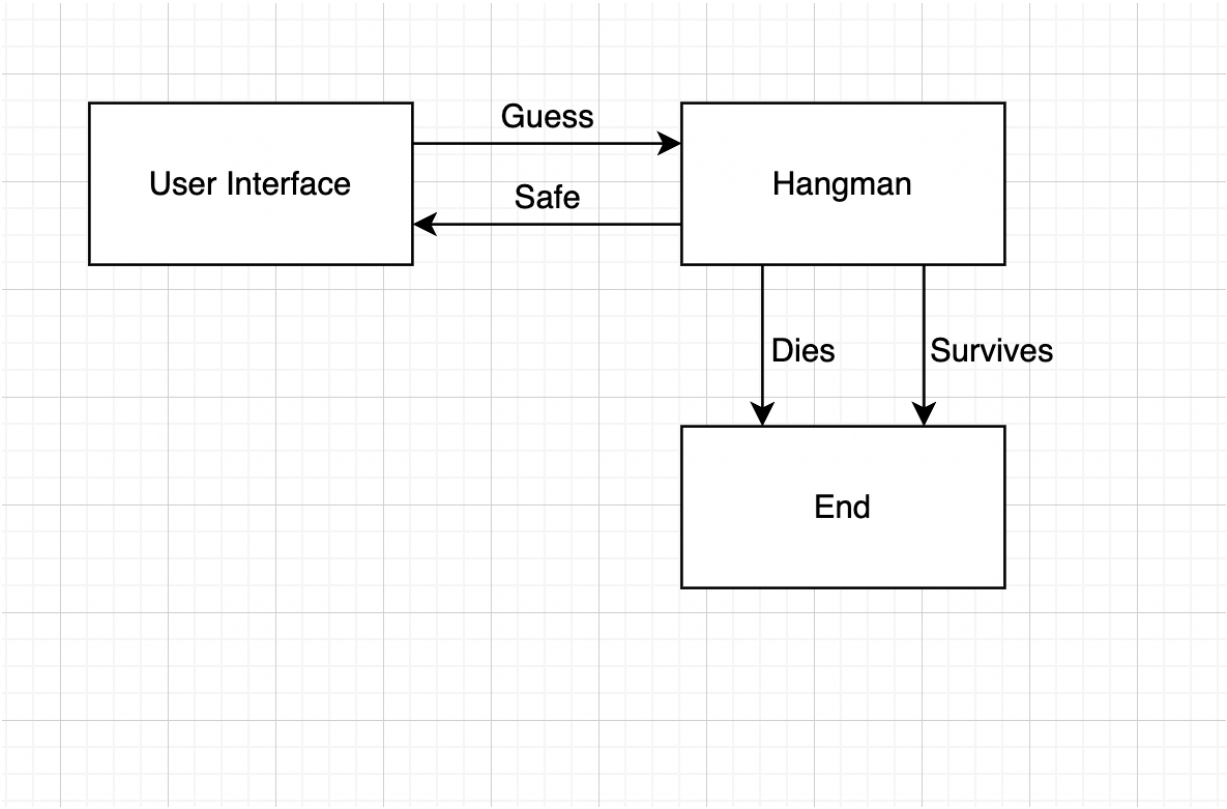
Snake



Flappy Bird

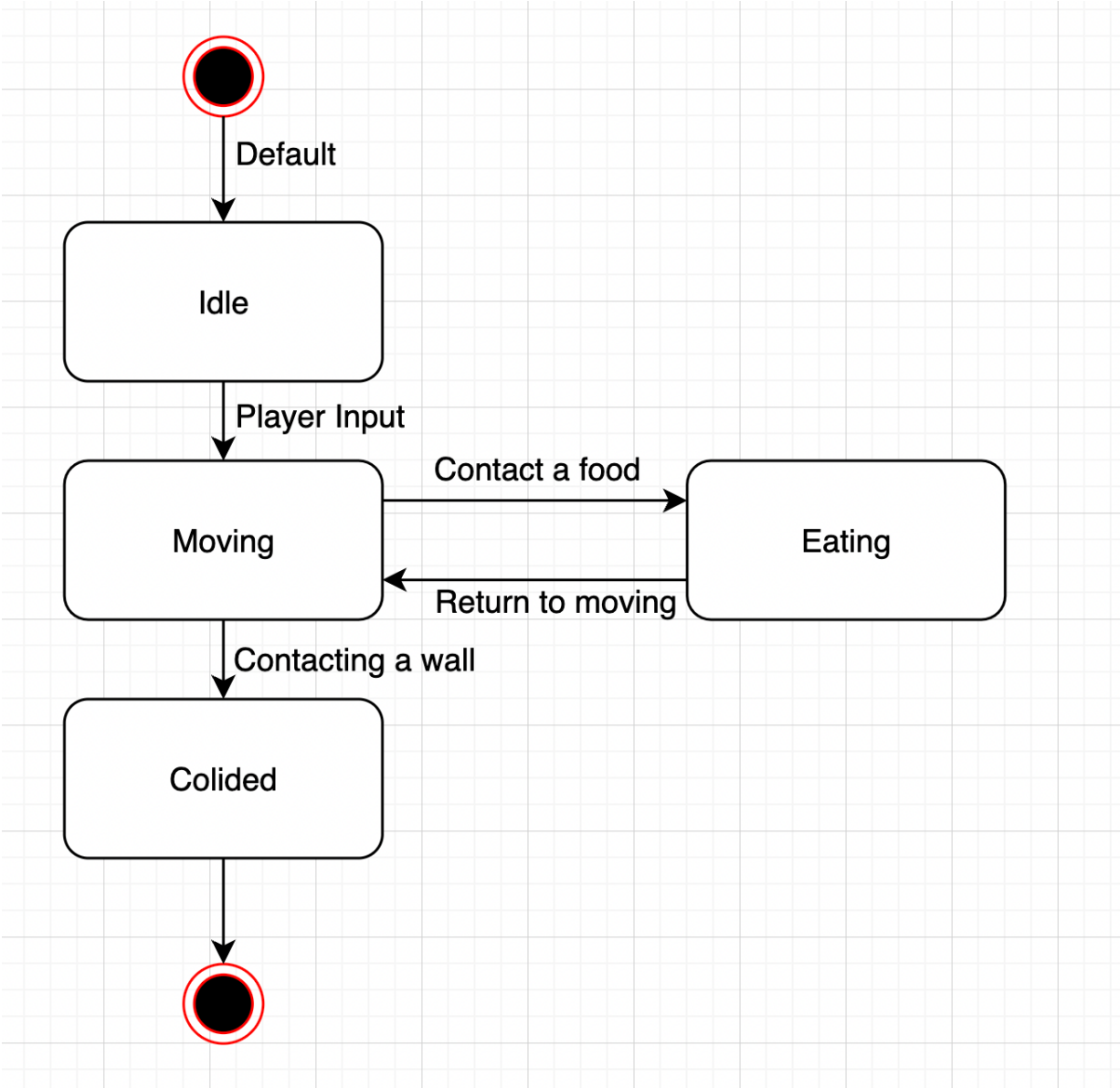


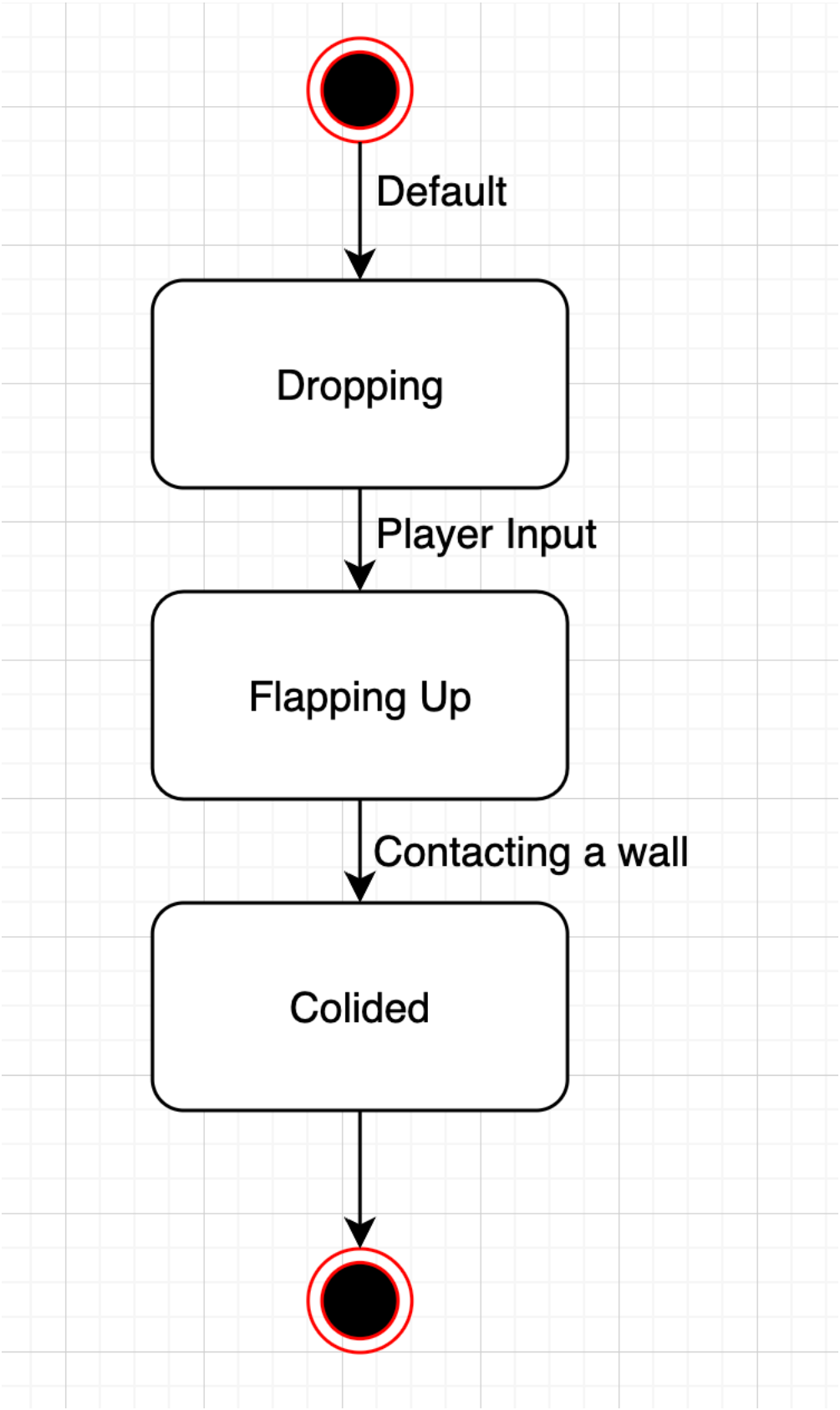
Hangman

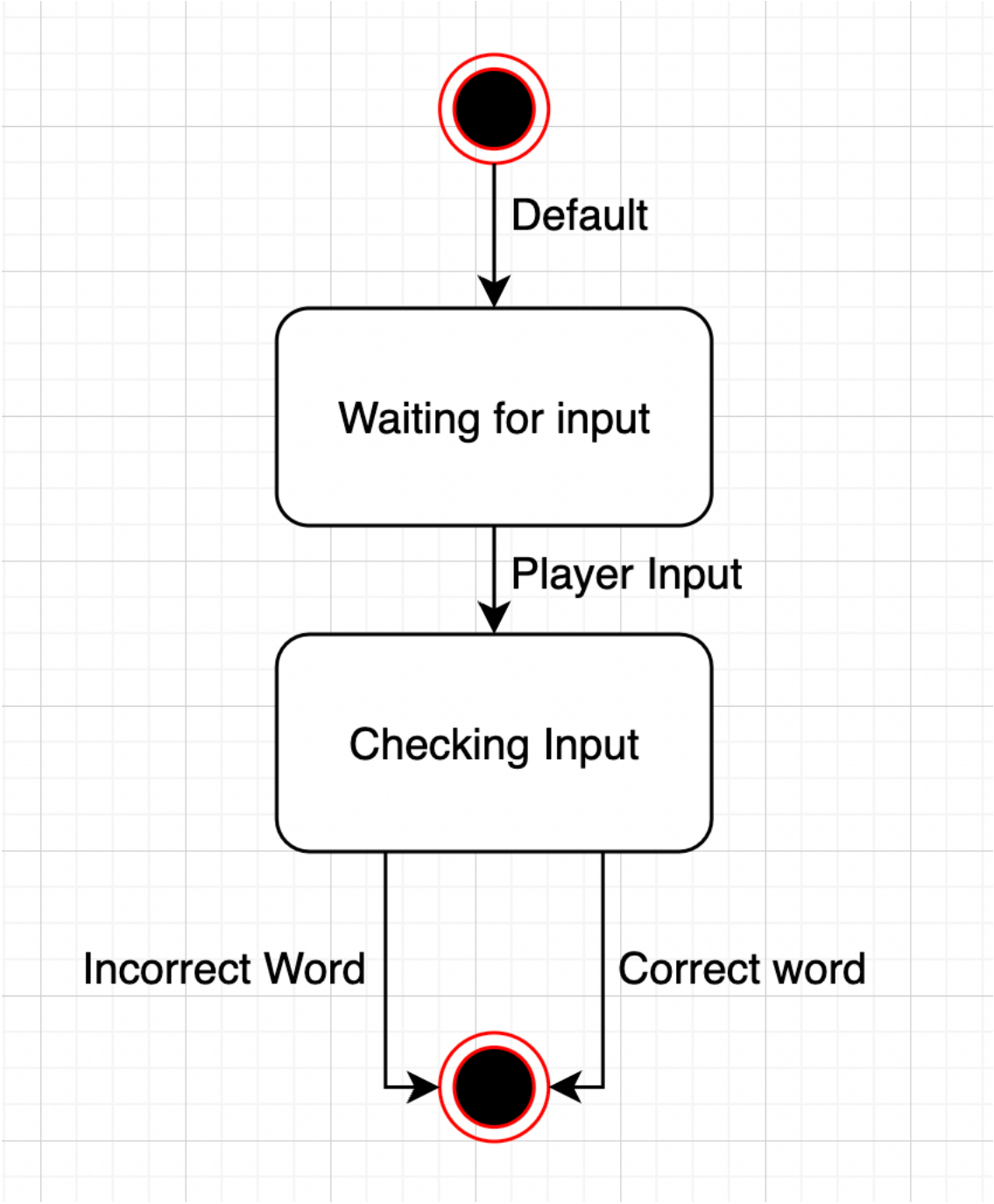


STATE CHART DIAGRAMS

Snake

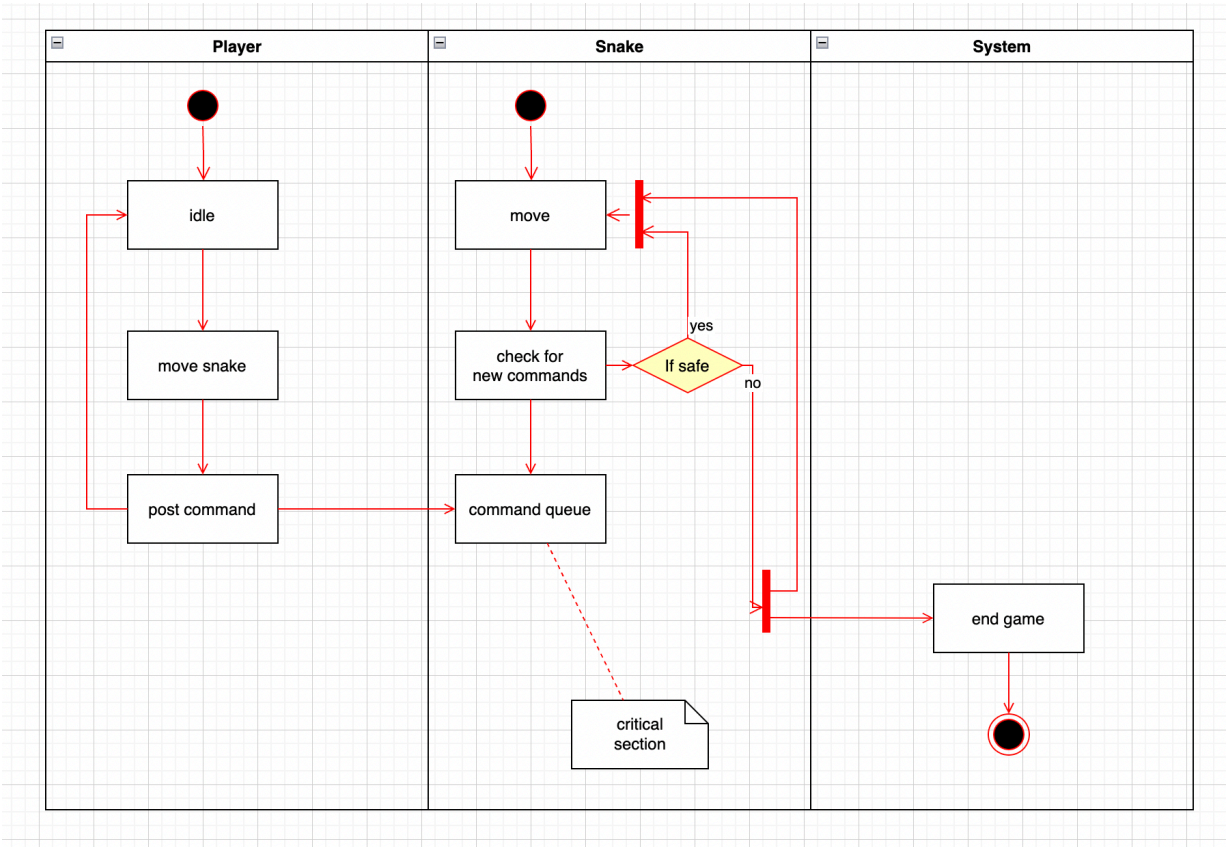



Default
Dropping
Player Input
Flapping Up
Contacting a wall
Colided

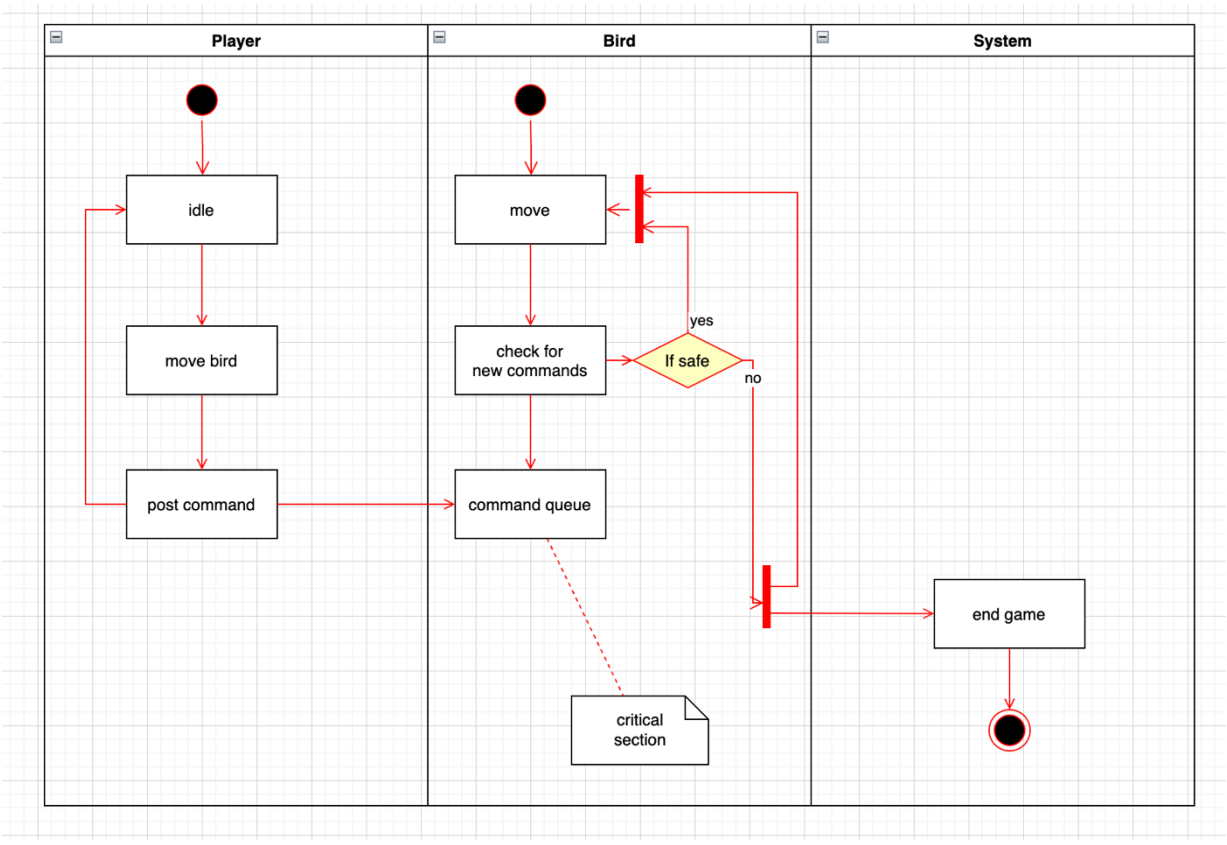

Default
Waiting for input
Player Input
Checking Input
Incorrect Word
Correct word

ACTIVITY DIAGRAMS

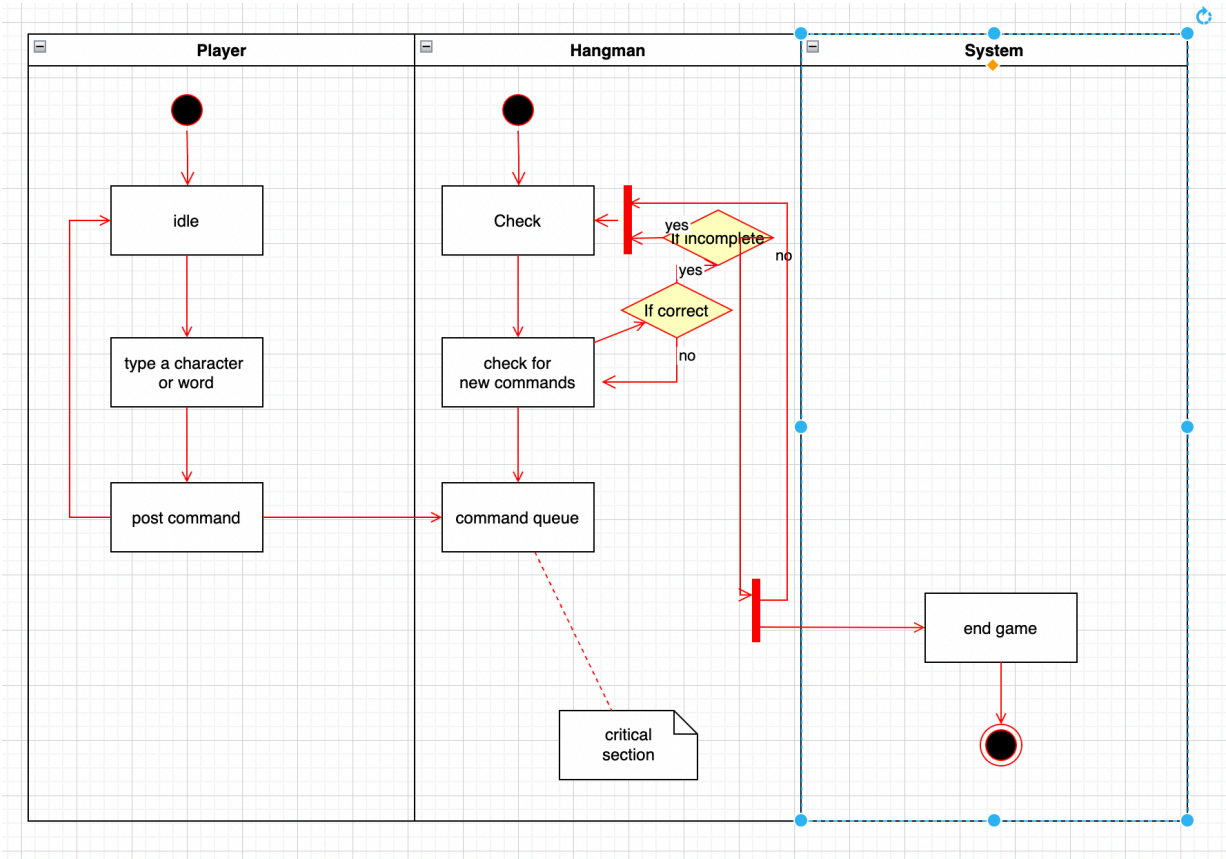
Snake



Flappy Bird

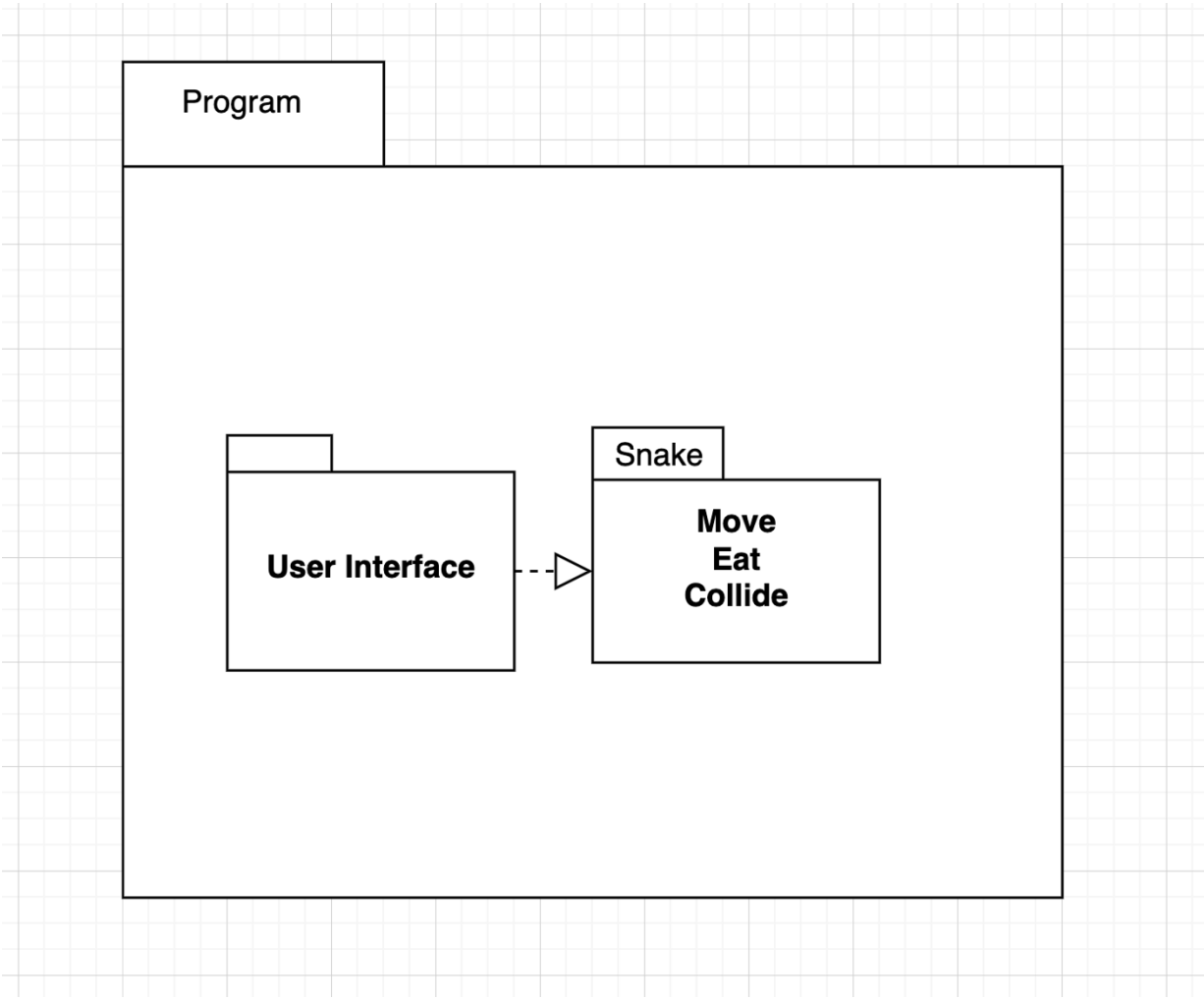


Hangman

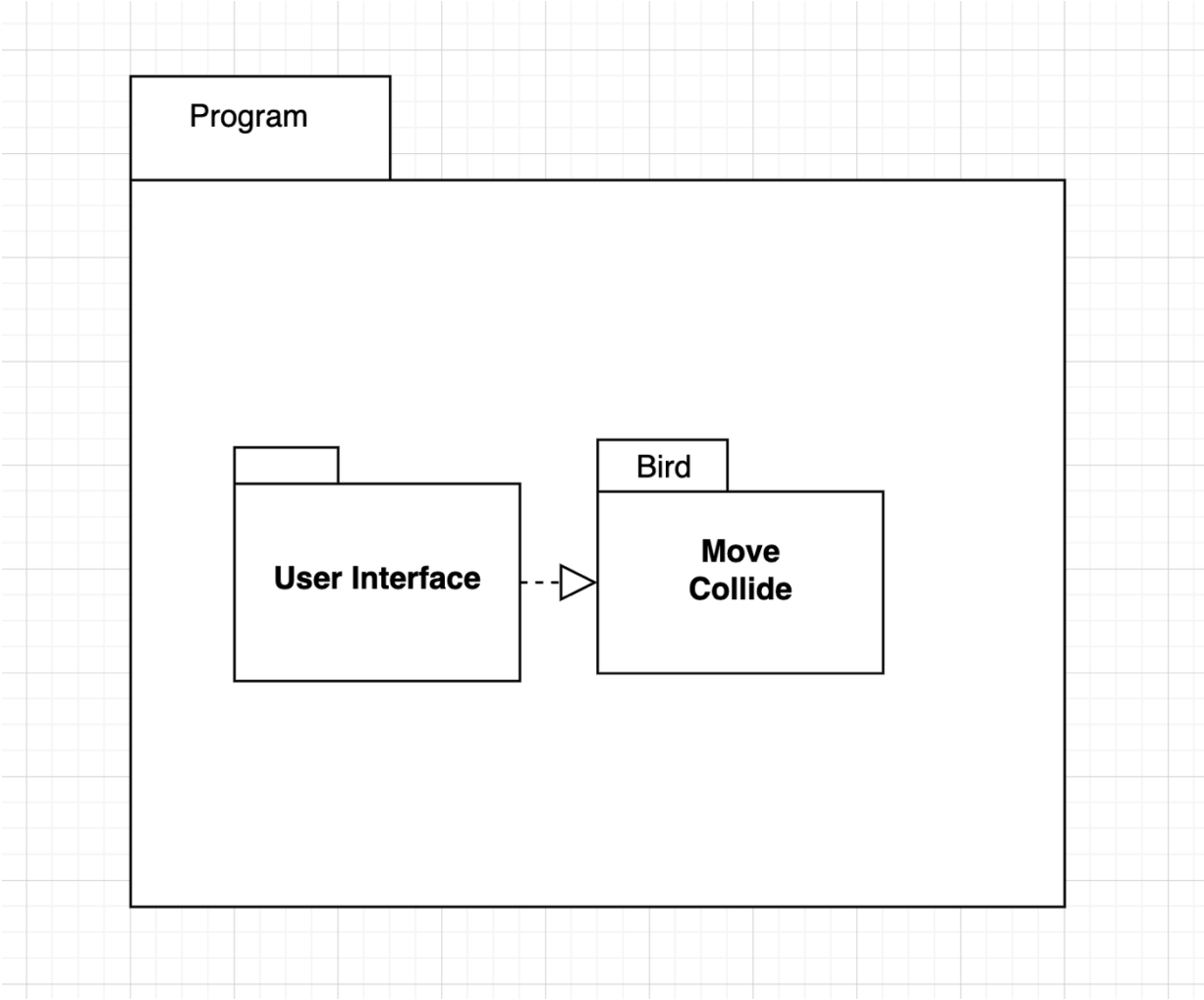

Player
Hangman
System
idle
type a character or word
post command
Check
If incomplete
yes
no
yes
If correct
no
check for new commands
command queue
critical section
end game

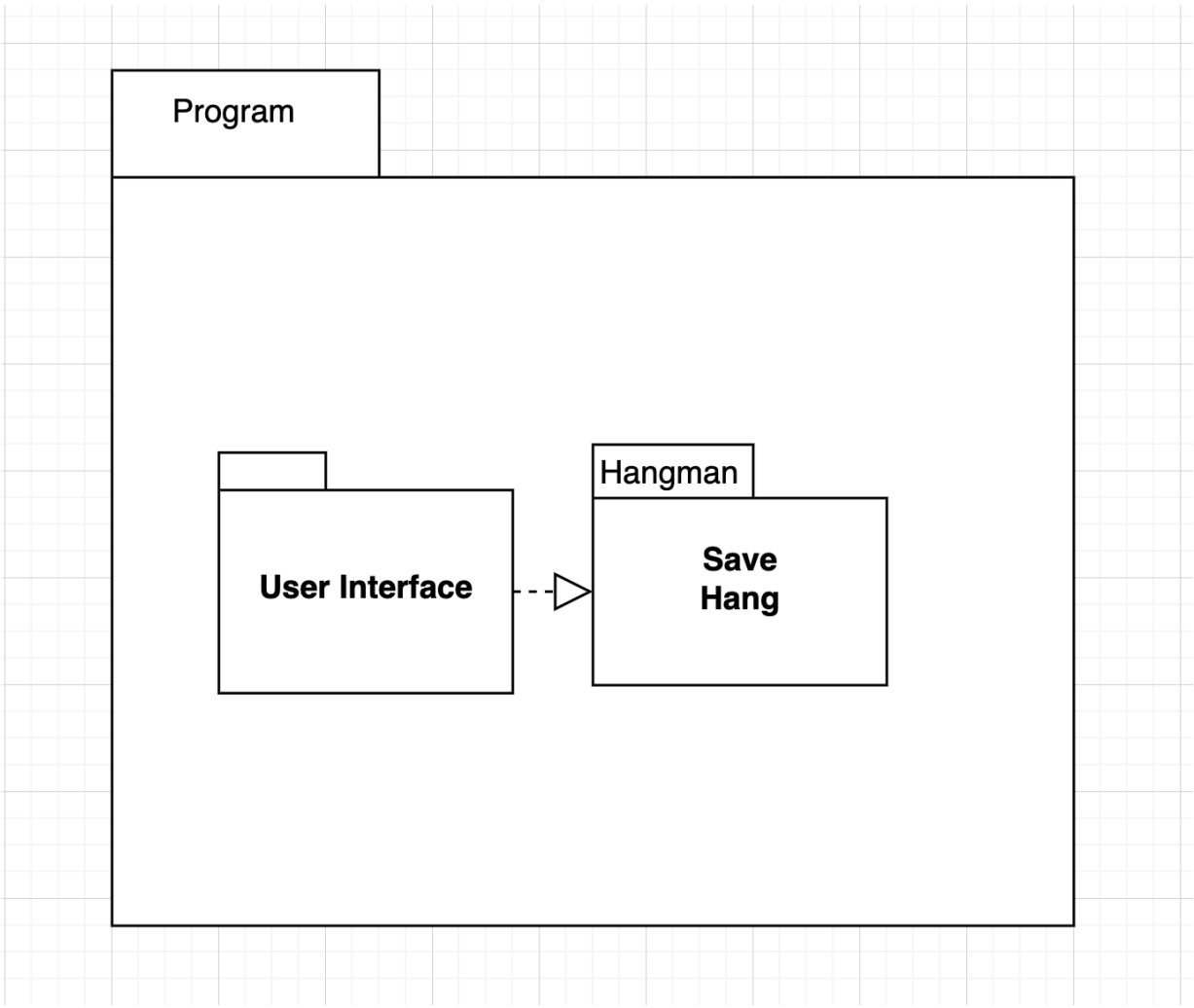
PACKAGE DIAGRAMS

Snake



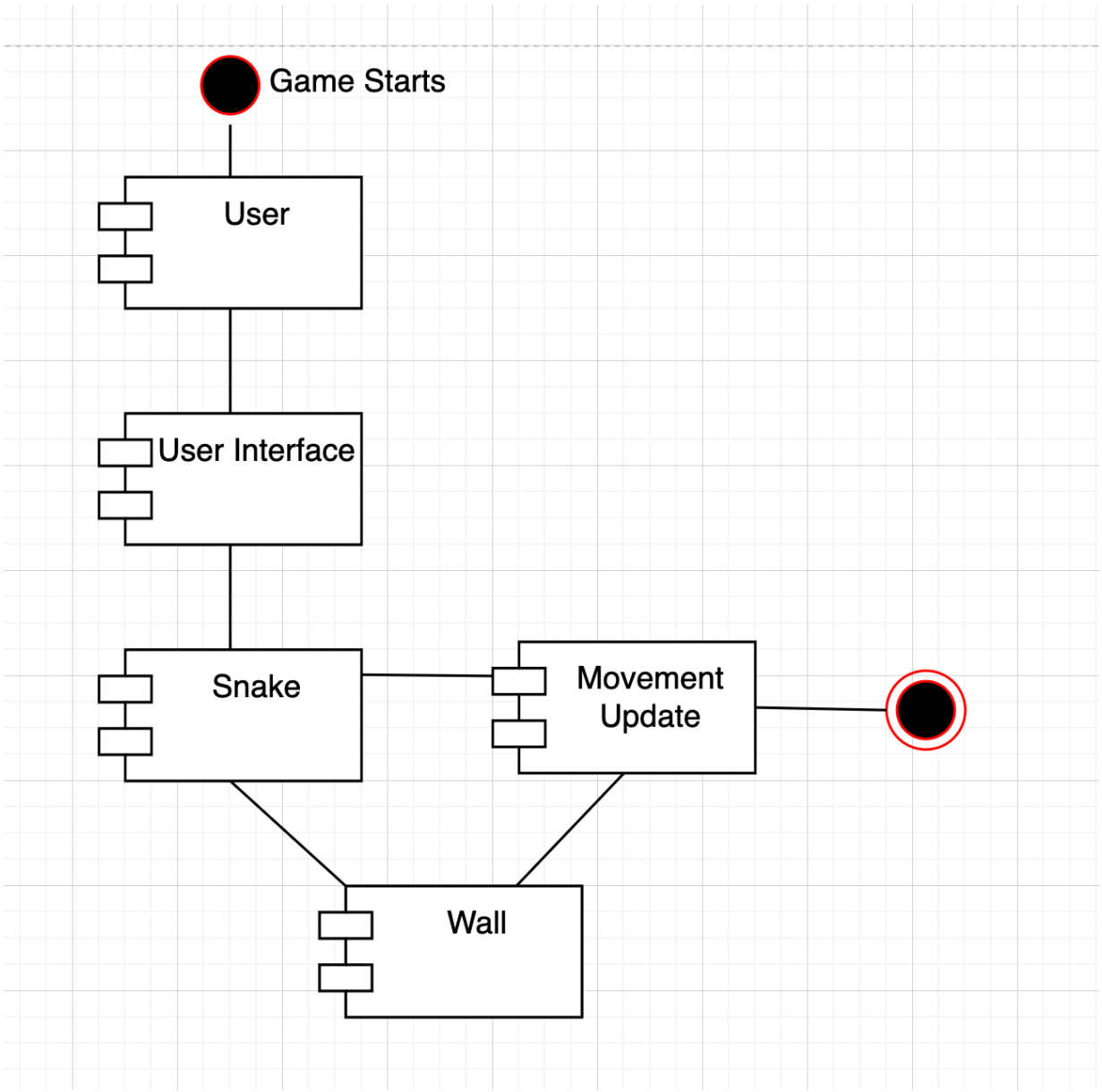
Flappy Bird



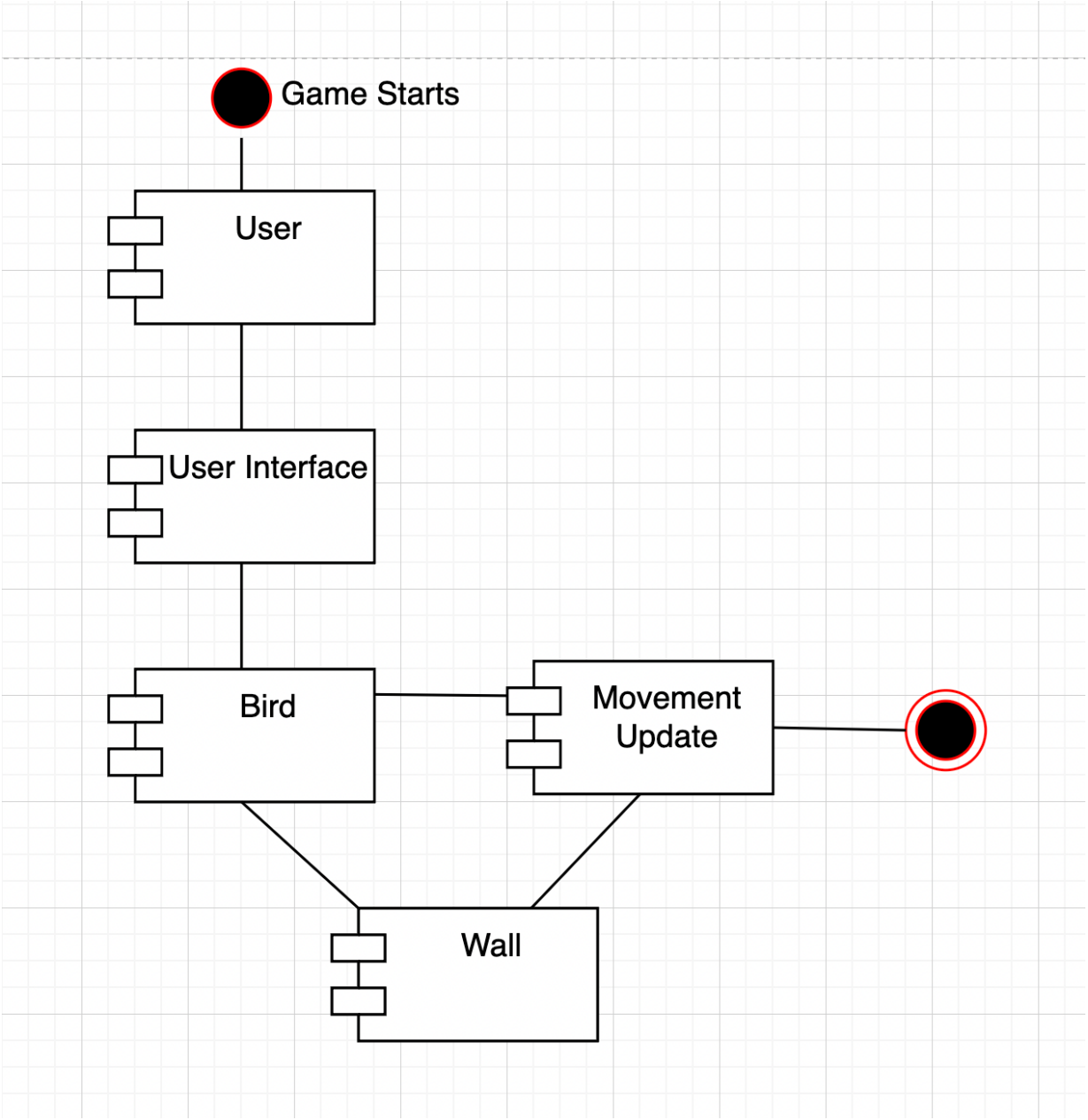

Program
User Interface
Hangman
Save
Hang

COMPONENT DIAGRAMS

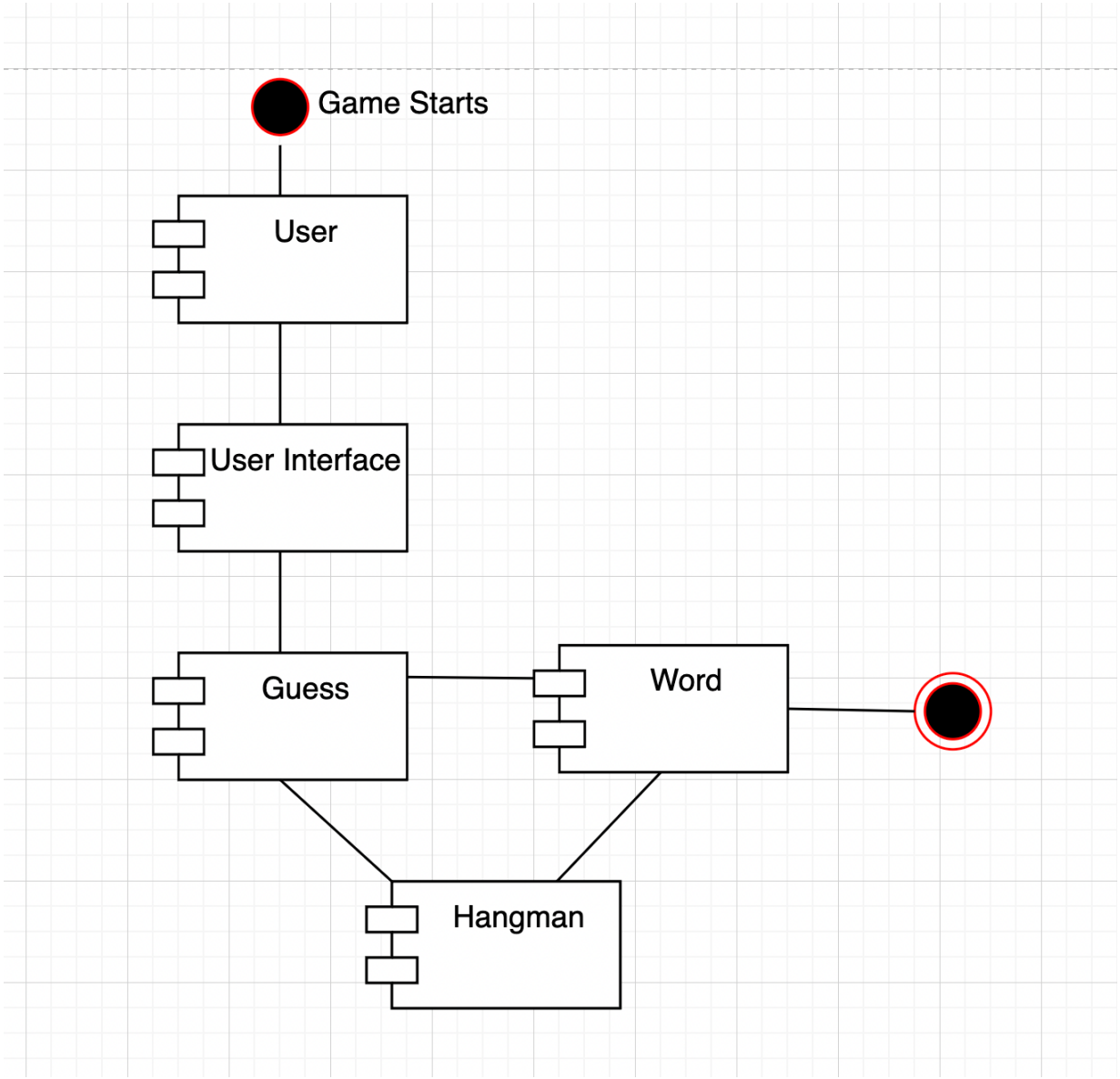
Snake



Flappy Bird

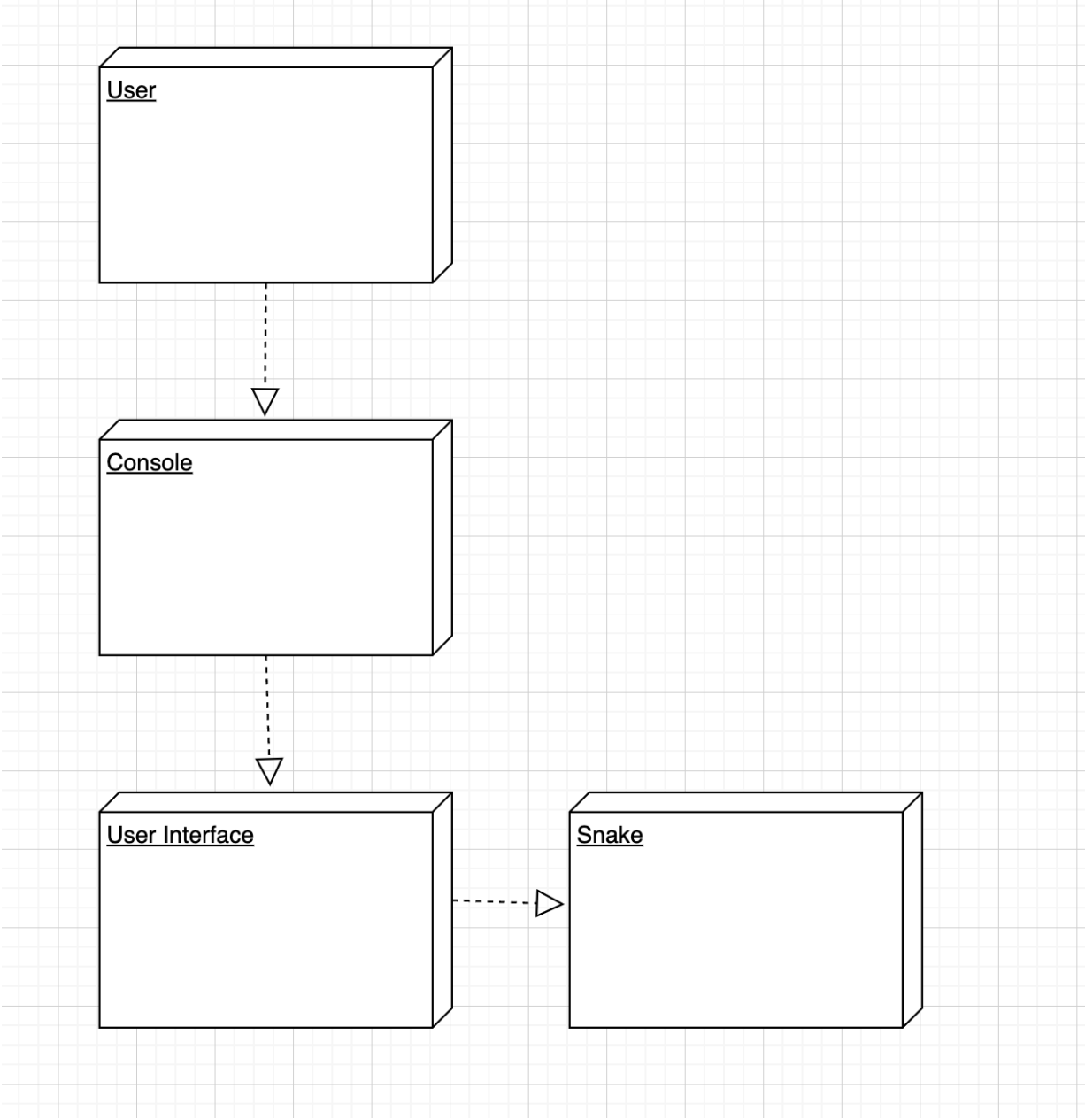


Hangman

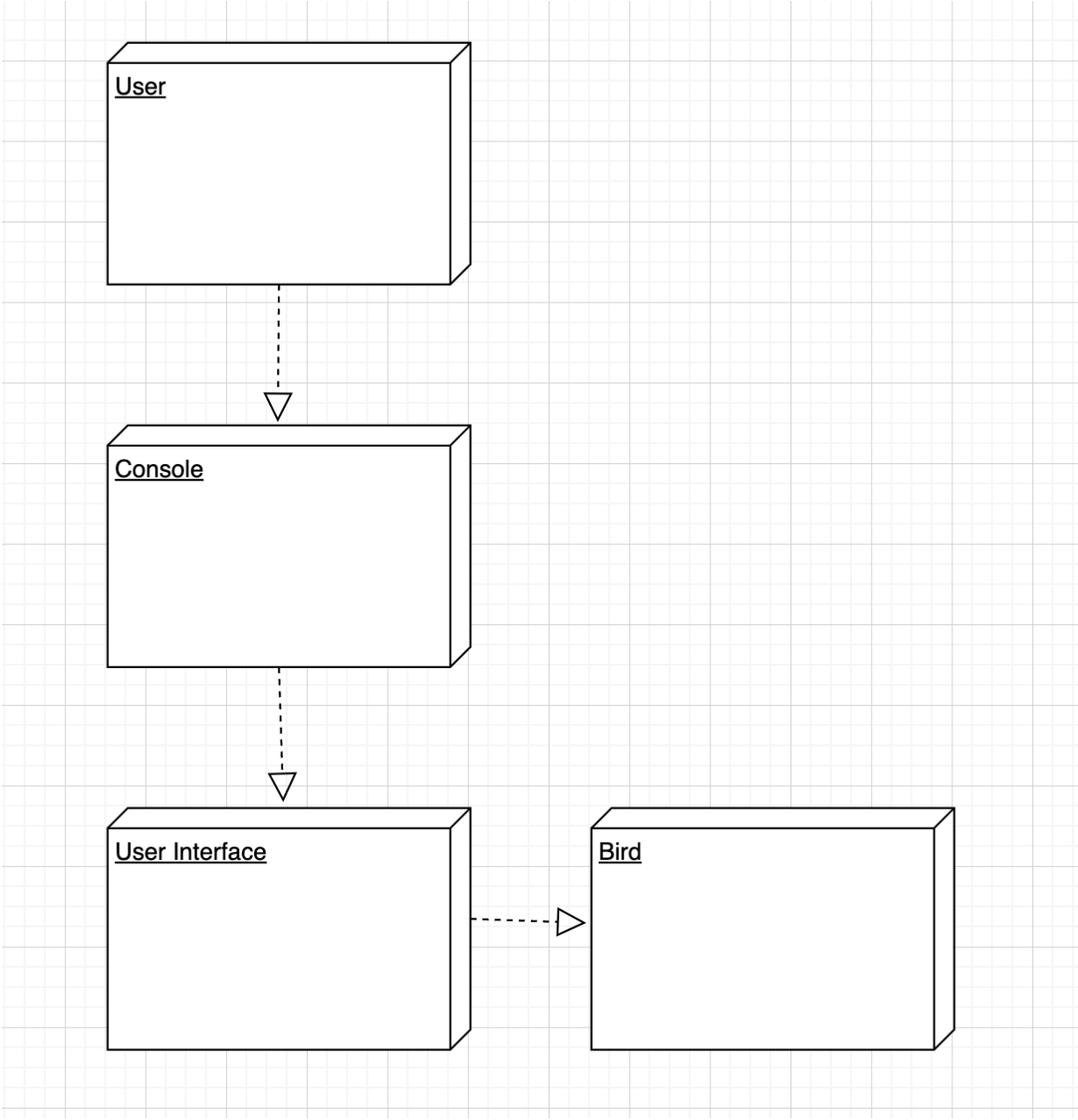


DEPLOYMENT DIAGRAMS

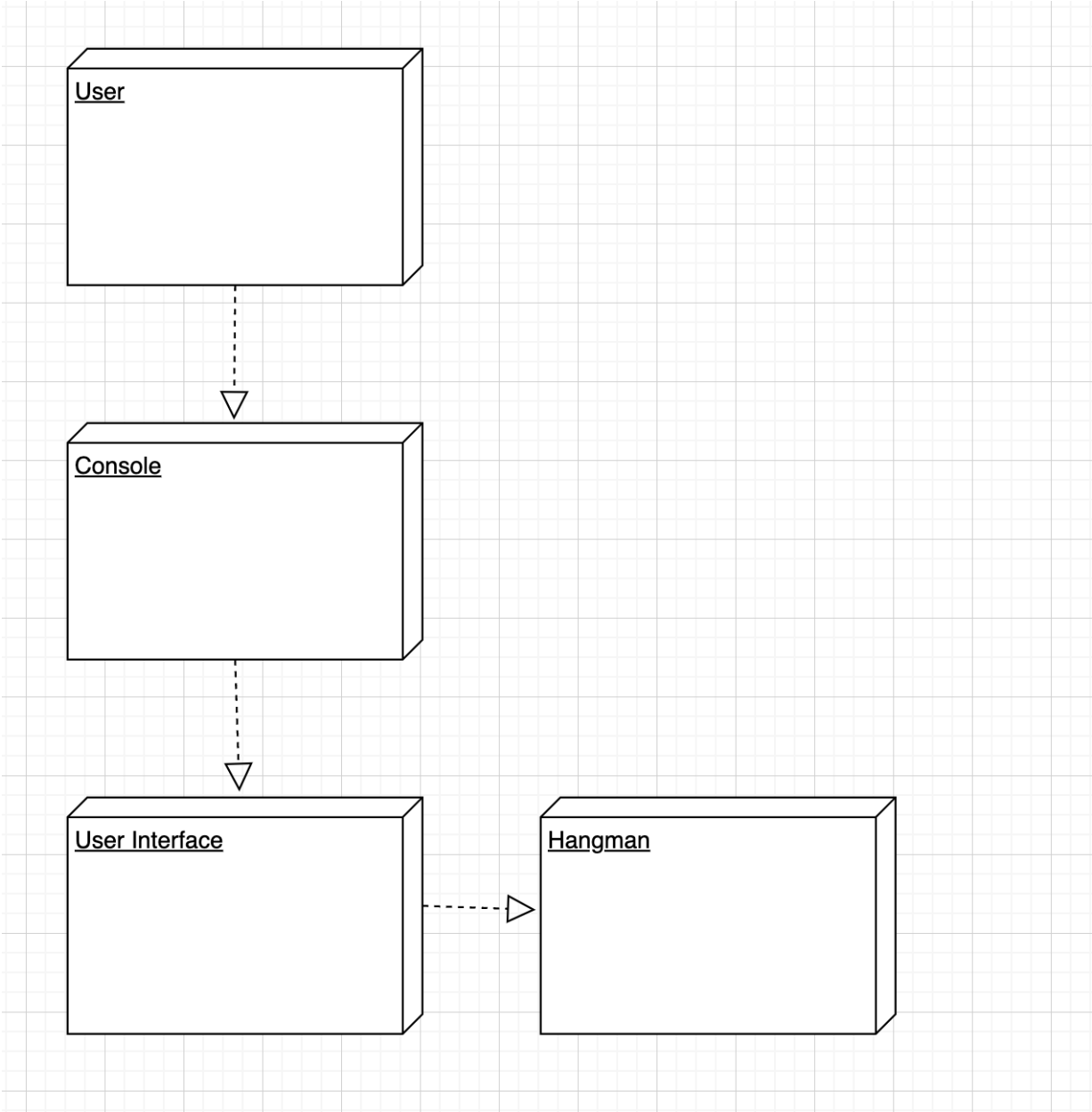
Snake



Flappy Bird



Hangman



CODES AND OUTPUT SCREENSHOTS

console.cpp

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <windows.h>


using namespace std;

int main() {
   system("color 05");
   string command;
   while (true) {
      system("cls");
      cout << "Enter a command ('snake', 'flappybird', 'hangman', or 'exit'): ";
      cin >> command;
      if (command == "snake")
      {
         system("g++ snake.cpp -o snake && snake");

      }

      else if (command == "flappybird")
      {
         system("g++ flappybird.cpp -o flappybird && flappybird");
      }

      else if (command == "hangman")
      {
         system("g++ hangman.cpp -o hangman && hangman");
      }

      else if (command == "exit")
      {
         break;
      }
      else
      {
         cout << "Invalid command." << endl;
      }
   }
   return 0;
}
```

Snake.cpp

```
#include <iostream>
#include <windows.h>
#include <stdlib.h>
#include <conio.h>
#include <fstream>


using namespace std;

//==== struct list;

typedef struct tailpos
{
    int x;              // x cordinate of the snake
    int y;              // y cordinate of the snake
    struct tailpos *next; // pointer to next node
    struct tailpos *prev; // pointer to previous node
} tail;

// d used to set the direction
int d = 4; // up = 1 , down = 2 , left =3 , right = 4;

/*
SNAKE CLASS
        contain variable of tailpos structure
        this class handle movement of the snake
        contain the draw function

*/
class snake
{
public:
    int wallsX, wallsY; // wall start position can be use to define wall
    int walleX, walleY; // wall end position

    int score; // keep score

    int foodx, foody; // position of the food

    HANDLE console_handle; // handle for the console
    COORD cur_cord;       // COORD struct to keep the coordinate
```

```
    /*
    tail struct ==>
            start = pointer to first part of body
            current = pointer to last node
            newtail = temp node for new tail

    */

    tail *start, *current, *newtail;
    snake();                // construct to initalise tail variable to null
    void insert(int x, int y); // insert the body of snake append to next node
    void draw();            // draw the snake
    void drawWall();        // draw the wall
    void move();            // control the movement
    bool collision();       //check snake collision with itself or wall
    void drawfood(int x = 0);  // draw food to new position if x==1
    void drawinit();        // inital setup like draw wall
    void labelDead();       // Draw when player is dead
    void menu();            // menu screen
    void help();            // menu for help
};

/*
=====================================
prototype for loop function
*/
void loop(snake &ob);

//=================================
// implement snake class
snake::snake()
{

    score = 0; // set the initial score

    start = NULL;                       // start node point to null
    current = NULL;                     // same
    newtail = NULL;                     // same
    console_handle = GetStdHandle(STD_OUTPUT_HANDLE); // getting console handle

    foodx = 12; // starting food positions
    foody = 14;

    wallsX = 2;
    wallsY = 2; // walls starting and ending position
    walleX = 70;
```

```
    walleY = 20; // dwar wall of 70x50

    cur_cord.X = 152;
    cur_cord.Y = 500;
    SetConsoleScreenBufferSize(console_handle, cur_cord); // setting up screen buffer
}
/*
this method is to insert new tail of snake at the current food position when snake eat it.

*/
void snake ::insert(int x, int y)
{
    // check if start is null
    if (start == NULL)
    {
        newtail = new tail;
        newtail->x = x;
        newtail->y = y;
        newtail->next = NULL;
        newtail->prev = NULL;
        start = newtail;
        current = newtail;
    }
    else // insert new node to start node
    {
        newtail = new tail;
        newtail->x = x;
        newtail->y = y;
        newtail->next = NULL;
        newtail->prev = current;
        current->next = newtail;
        current = newtail;
    }
}

/*
move :
    contain main logic to move the snake by incrementing and decrementing
    the X or Y cordinate
    position of first part is moved to the next part. copying is done in Reverse order that
    is from last to first .
    Last node position is updated with previous node position and at last. Start node Cordinate
    X or y is inc or dec.

*/
```

```
void snake::move()
{
    tail *tmp, *cur;

    tmp = current; // point tmp to last node
    //cur =start->next;

    while (tmp->prev != NULL)
    {
        tmp->x = tmp->prev->x; // copy val from revious to last
        tmp->y = tmp->prev->y;
        tmp = tmp->prev; // set tmp to previous node
    }
    /*
check for the value of d in order to change its direction
increment and decrement value of x , y
*/
    if (d == 1)
        start->y--;

    if (d == 2)
        start->y++;

    if (d == 3)
        start->x--;

    if (d == 4)
        start->x++;
}

/*
draw :
    draw snake part according to their position
    traverse the node from start to end
*/
void snake::draw()
{
    /*
cur_cord.X=10;
cur_cord.Y=5;

SetConsoleCursorPosition(console_handle,cur_cord);
cout << "use above statement to to set cursor position";
*/

    // putting score label
```

```
    cur_cord.X = 2;
    cur_cord.Y = 0;

    SetConsoleCursorPosition(console_handle, cur_cord);
    cout << "SCORE : " << score;

    tail *tmp, *last;
    tmp = start;
    last = current;

    while (tmp != NULL)
    {
        cur_cord.X = tmp->x;
        cur_cord.Y = tmp->y;
        SetConsoleCursorPosition(console_handle, cur_cord);
        //cout << "*" << cur_cord.X <<"-" <<  cur_cord.Y  ;
        cout << (char)219; // snake character
        tmp = tmp->next;
    }
    // remove tail
    cur_cord.X = last->x;
    cur_cord.Y = last->y;
    SetConsoleCursorPosition(console_handle, cur_cord);
    cout << ' ';
    //draw the food
    cur_cord.X = foodx;
    cur_cord.Y = foody;
    SetConsoleCursorPosition(console_handle, cur_cord);
    cout << (char)15; // food character
}

void snake::drawWall()
{
    /*
   for drawing wall you can use variables to store wall starting and ending position
   */
    //int size = sizeof(wall)/sizeof(wall[0][0]);
    // draw left column
    cur_cord.X = wallsX;
    for (int y = wallsY; y <= walleY; y++)
    {
        cur_cord.Y = y;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << '#';
    }
    // draw top row
```

```
    cur_cord.Y = wallsY;
    for (int x = wallsX; x <= walleX; x++)
    {
        cur_cord.X = x;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << '#';
    }
    // draw right column
    cur_cord.X = walleX;
    for (int y = wallsY; y <= walleY; y++)
    {
        cur_cord.Y = y;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << '#';
    }

    // draw bottom row
    cur_cord.Y = walleY;
    for (int x = wallsX; x <= walleX; x++)
    {
        cur_cord.X = x;
        SetConsoleCursorPosition(console_handle, cur_cord);
        cout << '#';
    }
}

void snake::drawfood(int x) // no use of int x here
{
    tail *tmp;
    tmp = start;
    if (x == 1) // draw new food
    {
        foodx = rand() % 2 + 39; // rand number between 2-39
        foody = rand() % 2 + 16; // it will better to use wall ends position walleX,Y
        /*check if food not created on snake body
            so we will loop through all snake node and check
            if snake body pos meet with food position
            if yes the generate new position else continue
            */
        while (tmp->next != NULL)
        {
            if (foodx == tmp->x && foody == tmp->y)
            {
                drawfood(1);
                cout << "drawn";
            }
```

```
            tmp = tmp->next;
        }
    }
}

void snake::drawinit()
{
    drawWall();
}

/*
decect if snake collided with it self or wall or food
*/
bool snake::collision()
{
    tail *tmp;
    tmp = start->next;
    //check collision with itself
    while (tmp->next != NULL)
    {
        if (start->x == tmp->x && start->y == tmp->y)
            return true;

        tmp = tmp->next;
    }
    //check collision with food
    if (start->x == foodx && start->y == foody)
    {
        insert(foodx, foody); // insert new tail
        drawfood(1);         // get new food position
        score++;             // update score
    }

    //check collision with wall
    //collision top row
    for (int x = wallsX; x <= walleX; x++)         // getting x coordinadte
    {                                   // y cordinate remain same y=0 because in row x increase in
same y
        if (start->x == x && start->y == wallsY) // getting y coordinte to get complete x,y
cordinate
        {
            return true;
        }
    }
    //collision left column
```

```
    for (int y = wallsY; y <= walleY; y++)
    {
        if (start->x == wallsX && start->y == y) // x same y chamge x=0
        {
            return true;
        }
    }
    //collision right column
    for (int y = wallsY; y <= walleY; y++)
    {
        if (start->x == walleX && start->y == y) // right column threfore x ending
point(same) with y changing
        {
            return true;
        }
    }
    //collision bottom row
    for (int x = wallsX; x <= walleX; x++)
    {
        if (start->x == x && start->y == walleY)
        {
            return true;
        }
    }

    return false; // return false no collison
}
/*
draw thing when player is dead
*/
void snake::labelDead()
{

    cur_cord.X = (walleX / 2);
    cur_cord.Y = (walleY / 2);

    SetConsoleCursorPosition(console_handle, cur_cord);

    cout << "YOU ARE DEAD\n";

    cur_cord.Y = (walleY / 2) + 1;
    SetConsoleCursorPosition(console_handle, cur_cord);
    cout << "YOUR HIGH SCORE IS " << score;

}
```

```
/*
        menu screen of the game
*/

void snake::menu()
{
    char word;
    ifstream iFile("menu.txt");
    word = iFile.get();
    while (iFile)
    {

        cout << word;
        word = iFile.get();
    }
    iFile.close();
    // get the menu
}

/*
help menu for snake
*/
void snake::help()
{
    char word;
    ifstream iFile("help.txt");
    word = iFile.get();
    while (iFile)
    {

        cout << word;
        word = iFile.get();
    }
    iFile.close();

    getch(); // wait for key to press
}

//===== main function
int main()
{
    system("color 05");
    // displaying the menu
    snake obc;
    obc.menu();
    switch (getch())
```

```
        {
        case 'z':
                system("CLS");
                loop(obc);
                break;
        case 'h':
                system("CLS");
                obc.help();
                system("CLS");
                main();
                break;
        case 'q':
                break;
        default:
                system("cls");
                main();
        }


        return 0;
}

// loop controls the game
void loop(snake &ob)
{
        ob.insert(10, 6); // first is head  snake drawn from first to last
        ob.insert(10, 7);
        ob.insert(10, 8);
        ob.insert(10, 9);

        ob.drawinit(); // this will just draw wall
        int dir = 1;
        while (1)
        {
                ob.draw();
                Sleep(200); // waiting time
                        //system("CLS");
                        //clearScreen();

                if (ob.collision()) // chek if collision with wall or itselfoccur
                {

                        ob.labelDead(); // do when snake is dead
                        system("pause");
```

```
            system("cls"); // clears the console screen
            //system("g++ console.cpp -o console && console");
            system("console.exe");
            break;



        }

        if (kbhit()) // check if keyboard key is pressed
        {
            switch (getch())
            {
            case 'w':
                d = 1;
                break;
            case 's':
                d = 2;
                break;
            case 'a':
                d = 3;
                break;
            case 'd':
                d = 4;
                break;
            case 'm':
                ob.insert(10, 7);
                break;
            }

        }

        ob.move();
    }
    int x;
    cin >> x;


}
```

Flappybird.cpp

```
#include<iostream>
#include<conio.h>
#include<dos.h>
#include<stdlib.h>
#include<string.h>
#include <windows.h>
#include <time.h>

#define SCREEN_WIDTH 90
#define SCREEN_HEIGHT 26
#define WIN_WIDTH 70
#define MENU_WIDTH 20
#define GAP_SIZE 7
#define PIPE_DIF 45

using namespace std;
//                  cout<<"";

HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
COORD CursorPosition;

int pipePos[3];
int gapPos[3];
int pipeFlag[3];
char bird[2][6] = { '/','-','-','o','\\',' ',
                                                '|','_','_','_',' ','>' };
int birdPos = 6;
int score = 0;

void gotoxy(int x, int y)
{
        CursorPosition.X = x;
        CursorPosition.Y = y;
        SetConsoleCursorPosition(console, CursorPosition);
}

void setcursor(bool visible, DWORD size)
{
        if(size == 0)
                size = 20;

        CONSOLE_CURSOR_INFO lpCursor;
        lpCursor.bVisible = visible;
```

```
	lpCursor.dwSize = size;
	SetConsoleCursorInfo(console,&lpCursor);
}

void drawBorder(){

	for(int i=0; i<SCREEN_WIDTH; i++){
		gotoxy(i,0); cout<<"";
		gotoxy(i,SCREEN_HEIGHT); cout<<"";
	}

	for(int i=0; i<SCREEN_HEIGHT; i++){
		gotoxy(0,i); cout<<"";
		gotoxy(SCREEN_WIDTH,i); cout<<"";
	}
	for(int i=0; i<SCREEN_HEIGHT; i++){
		gotoxy(WIN_WIDTH,i); cout<<"";
	}
}
void genPipe(int ind){
	gapPos[ind] = 3 + rand()%14;
}
void drawPipe(int ind){
	if( pipeFlag[ind] == true ){
		for(int i=0; i<gapPos[ind]; i++){
			gotoxy(WIN_WIDTH-pipePos[ind],i+1); cout<<"***";
		}
		for(int i=gapPos[ind]+GAP_SIZE; i<SCREEN_HEIGHT-1; i++){
			gotoxy(WIN_WIDTH-pipePos[ind],i+1); cout<<"***";
		}
	}
}
void erasePipe(int ind){
	if( pipeFlag[ind] == true ){
		for(int i=0; i<gapPos[ind]; i++){
			gotoxy(WIN_WIDTH-pipePos[ind],i+1); cout<<"   ";
		}
		for(int i=gapPos[ind]+GAP_SIZE; i<SCREEN_HEIGHT-1; i++){
			gotoxy(WIN_WIDTH-pipePos[ind],i+1); cout<<"   ";
		}
	}
}

void drawBird(){
	for(int i=0; i<2; i++){
		for(int j=0; j<6; j++){
```

```
                        gotoxy(j+2,i+birdPos); cout<<bird[i][j];
                }
        }
}
void eraseBird(){
        for(int i=0; i<2; i++){
                for(int j=0; j<6; j++){
                        gotoxy(j+2,i+birdPos); cout<<" ";
                }
        }
}

int collision(){
        if( pipePos[0] >= 61  ){
                if( birdPos<gapPos[0] || birdPos >gapPos[0]+GAP_SIZE ){
//                      cout<< " HIT ";
//                      getch();
                        return 1;
                }
        }
        return 0;
}
void debug(){
//      gotoxy(SCREEN_WIDTH + 3, 4); cout<<"Pipe Pos: "<<pipePos[0];

}
void gameover(){
        system("cls");
        cout<<endl;
        cout<<"\t\t--------------------------"<<endl;
        cout<<"\t\t-------- Game Over -------"<<endl;
        cout<<"\t\t--------------------------"<<endl<<endl;
        cout<<"\t\tPress any key to go back to menu.";
        getch();
}
void updateScore(){
        gotoxy(WIN_WIDTH + 7, 5);cout<<"Score: "<<score<<endl;
}

void instructions(){

        system("cls");
        cout<<"Instructions";
        cout<<"\n----------------";
        cout<<"\n Press spacebar to make bird fly";
        cout<<"\n\nPress any key to go back to menu";
```

```
	getch();
}

void play(){

	birdPos = 6;
	score = 0;
	pipeFlag[0] = 1;
	pipeFlag[1] = 0;
	pipePos[0] = pipePos[1] = 4;

	system("cls");
	drawBorder();
	genPipe(0);
	updateScore();

	gotoxy(WIN_WIDTH + 5, 2);cout<<"FLAPPY BIRD";
	gotoxy(WIN_WIDTH + 6, 4);cout<<"----------";
	gotoxy(WIN_WIDTH + 6, 6);cout<<"----------";
	gotoxy(WIN_WIDTH + 7, 12);cout<<"Control ";
	gotoxy(WIN_WIDTH + 7, 13);cout<<"-------- ";
	gotoxy(WIN_WIDTH + 2, 14);cout<<" Spacebar = jump";

	gotoxy(10, 5);cout<<"Press any key to start";
	getch();
	gotoxy(10, 5);cout<<"                    ";

	while(1){

		if(kbhit()){
			char ch = getch();
			if(ch==32){
				if( birdPos > 3 )
					birdPos-=3;
			}
			if(ch==27){
				break;
			}
		}

		drawBird();
		drawPipe(0);
		drawPipe(1);
		debug();
		if( collision() == 1 ){
			gameover();
```

```
                return;
            }
            Sleep(100);
            eraseBird();
            erasePipe(0);
            erasePipe(1);
            birdPos += 1;

            if( birdPos > SCREEN_HEIGHT - 2 ){
                gameover();
                return;
            }

            if( pipeFlag[0] == 1 )
                pipePos[0] += 2;

            if( pipeFlag[1] == 1 )
                pipePos[1] += 2;

            if( pipePos[0] >= 40 && pipePos[0] < 42 ){
                pipeFlag[1] = 1;
                pipePos[1] = 4;
                genPipe(1);
            }
            if( pipePos[0] > 68 ){
                score++;
                updateScore();
                pipeFlag[1] = 0;
                pipePos[0] = pipePos[1];
                gapPos[0] = gapPos[1];
            }

        }

}

int main()
{
        system("color 05");
        setcursor(0,0);
        srand( (unsigned)time(NULL));

//      play();

        do{
            system("cls");
```

```
                gotoxy(10,5); cout<<" -------------------------- ";
                gotoxy(10,6); cout<<" |      Flappy Bird       | ";
                gotoxy(10,7); cout<<" --------------------------";
                gotoxy(10,9); cout<<"1. Start Game";
                gotoxy(10,10); cout<<"2. Instructions";
                gotoxy(10,11); cout<<"3. Quit";
                gotoxy(10,13); cout<<"Select option: ";
                char op = getche();

                if( op=='1') play();
                else if( op=='2') instructions();
                else if( op=='3') exit(0);

        }while(1);

        return 0;

}
```

Hangman.cpp

```
#include <iostream>
#include <cstdlib>
#include<ctime>
#include <string>
using namespace std;

int NUM_TRY=3;
int checkGuess (char, string, string&);
void main_menu();
string message = "Play!";


int main(int argc, char *argv[])
{
        string name;
        char letter;
        string month;


   string months[] =
        {
                "january",
                "february",
                "march",
```

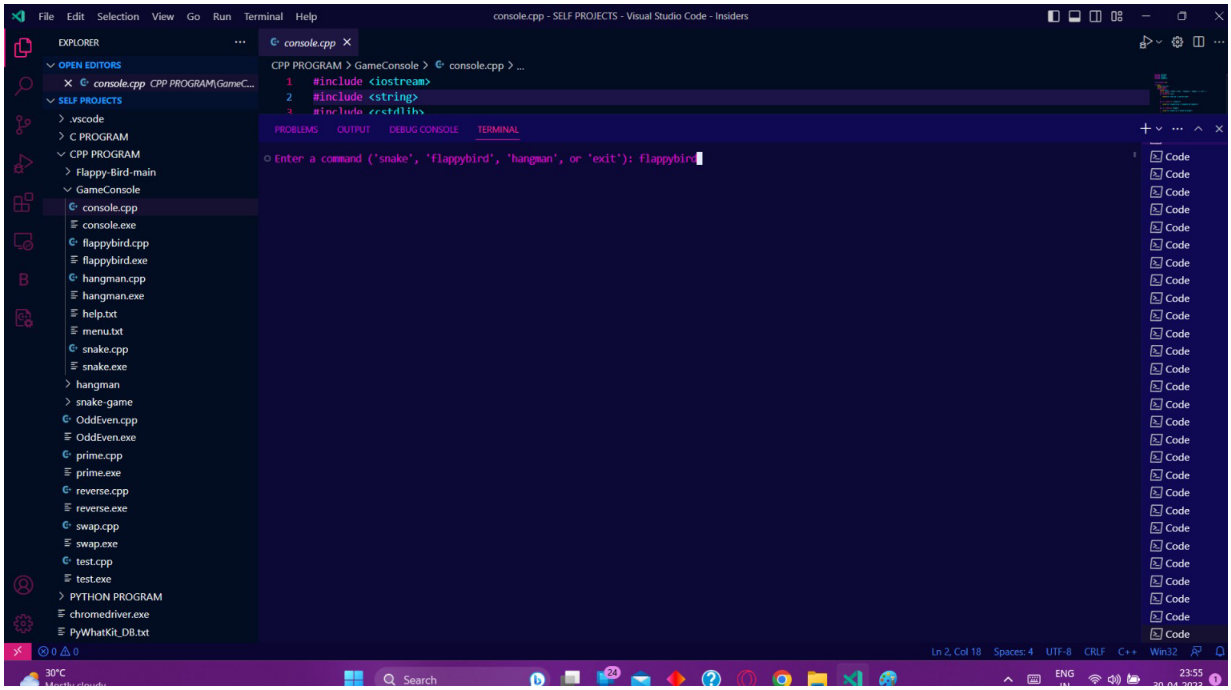
```
        "april",
        "may",
        "june",
        "july",
        "august",
        "september",
        "october",
        "november",
        "december"
};

srand(time(NULL));
int n=rand()% 12;
month=months[n];


string hide_m(month.length(),'X');




while (NUM_TRY!=0)
{
        main_menu();
        cout << "\n\n\t\t\t" << hide_m;
        cout << "\n\n\t\t\t\tGuess a letter: ";
        cin >> letter;

        if (checkGuess(letter, month, hide_m)==0)
        {
                message = "Incorrect letter.";
                NUM_TRY = NUM_TRY - 1;
        }
        else
        {
                message = "NICE! You guess a letter";
        }



        if (month==hide_m)
        {
                message = "Congratulations! You got it!";
                main_menu();
                cout << "\n\t\t\t\tThe month is : " << month << endl;
                break;
```

```
                }
        }
        if(NUM_TRY == 0)
        {
                message = "NOOOOOOO!...you've been hanged.";
                main_menu();
                cout << "\n\t\t\t\tThe month was : " << month << endl;
        }
        cin.ignore();
        cin.get();
        return 0;
}


int checkGuess (char guess, string secretmonth, string &guessmonth)
{
        int i;
        int matches=0;
        int len=secretmonth.length();
        for (i = 0; i< len; i++)
        {

                if (guess == guessmonth[i])
                        return 0;

                if (guess == secretmonth[i])
                {
                        guessmonth[i] = guess;
                        matches++;
                }
        }
        return matches;
}

void main_menu()
{
         system("color 05");
           system("cls");
            cout<<"\t\t\t\t*\t*";

  cout<<"\t\t\t\t**\t**";
   cout<<"\t\t\t\t***\t***";
   cout<<"\t\t\t\t****\t****";
   cout<<"\t\t\t\t*****\t*****";
    cout<<"\t\t\t\t******\t******";
   cout<<"\t\t\t\t*******\t*******";
```

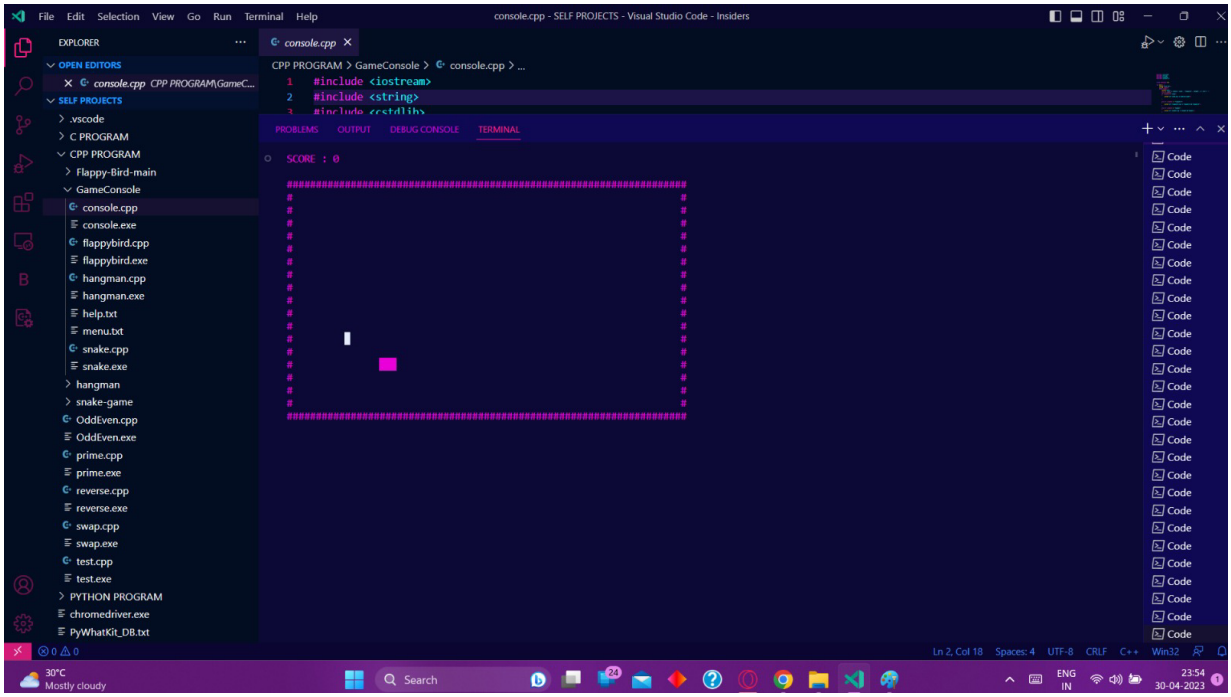
```
   cout<<"\t\t\t\t*******\t*******";
   cout<<"\t\t\t\t******\t******";
   cout<<"\t\t\t\t*****\t*****";
   cout<<"\t\t\t\t****\t****";
   cout<<"\t\t\t\t***\t***";
   cout<<"\t\t\t\t**\t**";
   cout<<"\t\t\t\t*\t*";


cout<<"\t\t@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@";
                cout<<"\n\t\t\t\tHangman Game!";
                                                cout << "\n\t\tYou have " << NUM_TRY << " tries to try and
guess the month.";
                                                cout<<"\n\n\t\t\t\t"+message;

cout<<"\n\t\t@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@\n";

}
```

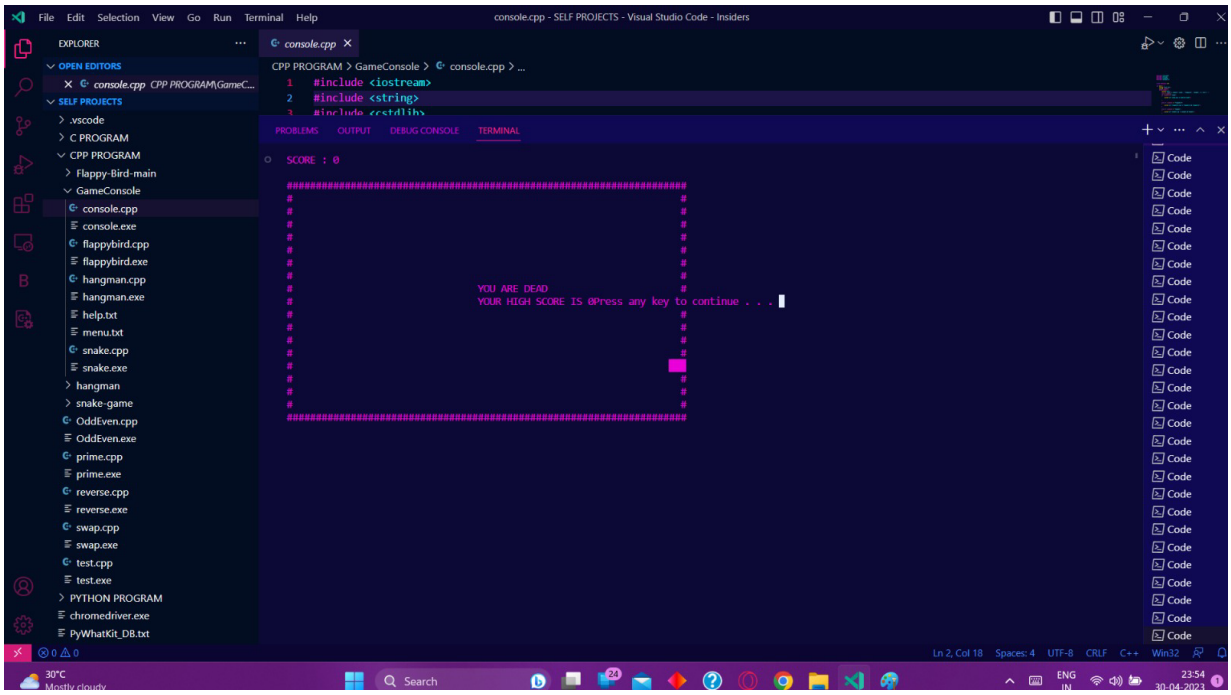
OUTPUT

CONSOLE

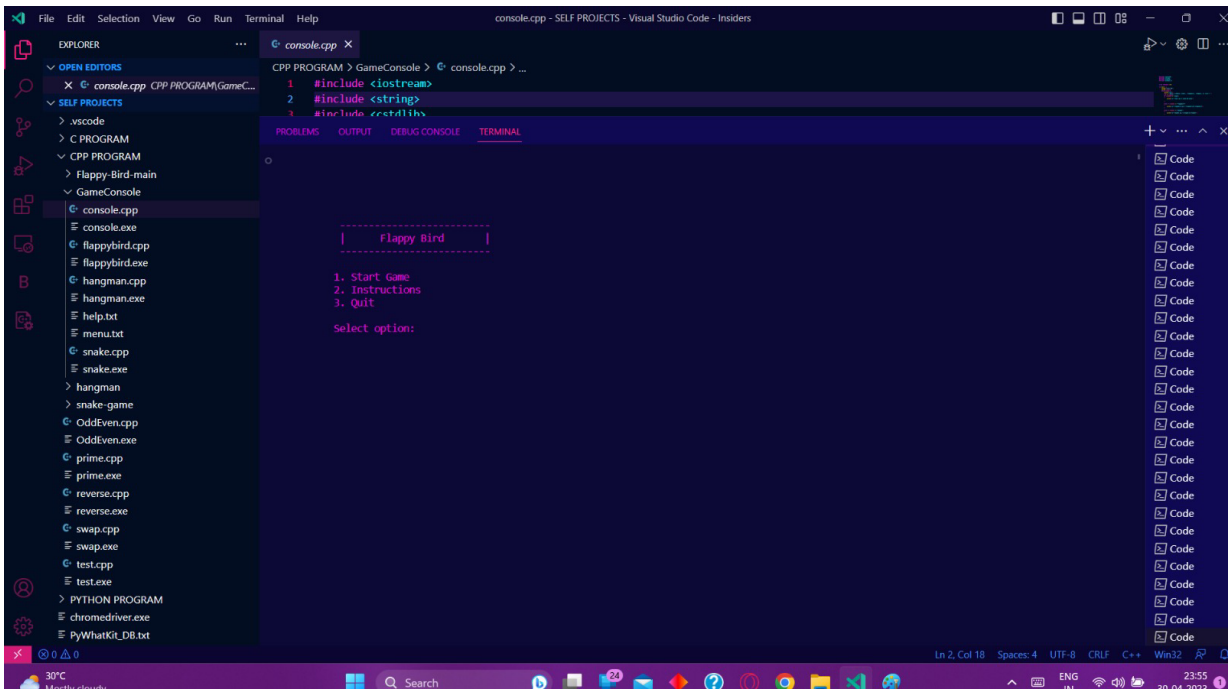


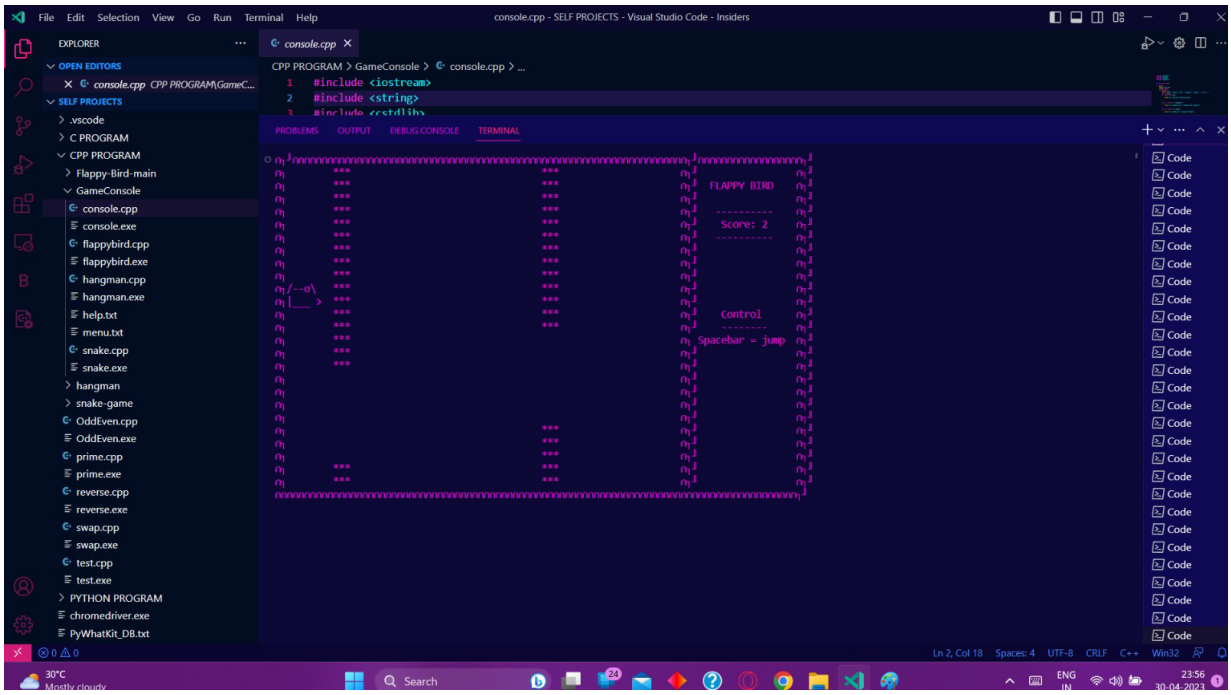
SNAKE



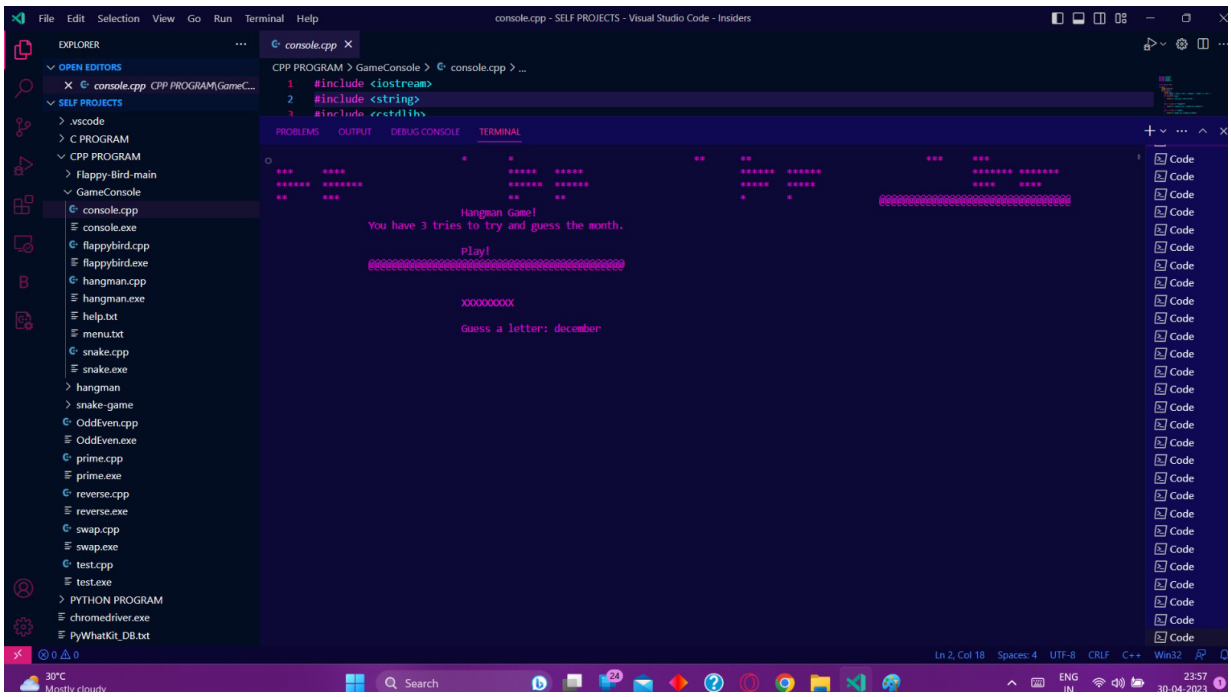

SCORE : 0
YOU ARE DEAD
YOUR HIGH SCORE IS 0Press any key to continue . . .

FLAPPY BIRD


Flappy Bird
1. Start Game
2. Instructions
3. Quit
Select option:


FLAPPY BIRD
Score: 2
Control
Spacebar = jump

HANGMAN


Hangman Game!
You have 3 tries to try and guess the month.
Play!
XXXXXXXXX
Guess a letter: december

CONCLUSION

In conclusion, the design and implementation of a gaming console that can access multiple gaming programs in C++ is a challenging task that requires careful consideration of several factors. These factors include the programming language used, the user interface design, the system performance, and the compatibility of the gaming programs.

Choosing the right programming language is crucial for creating a gaming console that can provide high performance and low-level control over hardware. C++ is an excellent choice as it is a high-performance language that is well-suited for creating games and other graphics-intensive applications. Additionally, many popular game engines and libraries are written in C++, making it easier to integrate with existing gaming programs.

The user interface design is equally important for creating a gaming console that is user-friendly and intuitive. The console must provide a clear and concise interface that allows users to easily navigate between games and perform other actions, such as saving and loading game progress. The interface should be intuitive and responsive, with clear instructions for users. Additionally, the console should provide feedback to users, such as notifications when a game is loading or when an action is completed.

System performance is another crucial factor that determines the success of a gaming console. The console must have sufficient memory and processing power to handle the demands of modern games. It should also be able to run multiple gaming programs simultaneously without causing lag or other performance issues. Thorough testing under different conditions, such as running multiple games at once or running graphics-intensive games, is necessary to ensure optimal performance.

The compatibility of the gaming programs is another essential consideration. The gaming console must be able to access and run multiple gaming programs written in C++. This requires careful attention to the file format, libraries, and dependencies used by each game. Additionally, the console should be able to handle different input devices, such as keyboards, mice, and game controllers, to ensure that users can play games using their preferred input method.

In conclusion, the design and implementation of a gaming console that can access multiple gaming programs in C++ is a complex process that requires careful planning and attention to detail. The success of the console ultimately depends on its ability to meet the needs and expectations of its users. By choosing the right programming language, creating a user-friendly interface, ensuring optimal system performance, and ensuring compatibility with multiple gaming programs, it is possible to create a gaming console that provides a seamless and immersive gaming experience to users. Thorough testing and user feedback are also essential for identifying and addressing any issues or limitations of the console to ensure its continued success.

RESULTS

The design and implementation of a gaming console that can access multiple gaming programs in C++ can be a challenging task. However, with careful planning and attention to detail, it is possible to create a system that is both functional and user-friendly.

To ensure that the gaming console works as intended and provides a seamless gaming experience to users, several factors must be considered. These include the programming language used, the user interface design, the system performance, and the compatibility of the gaming programs.

The use of C++ as the programming language for the gaming console is a good choice. C++ is a high-performance language that is well-suited for creating games and other graphics-intensive applications. It also provides low-level control over hardware, making it ideal for creating a gaming console. Additionally, many popular game engines and libraries, such as Unreal Engine and OpenGL, are written in C++, which makes it easier to integrate with existing gaming programs.

The user interface design is another crucial factor that determines the success of the gaming console. The console must provide a user-friendly interface that allows users to easily navigate between games and perform other actions, such as saving and loading game progress. The interface should be intuitive and responsive, with clear and concise instructions for users. Additionally, the console should provide feedback to users, such as notifications when a game is loading or when an action is completed.

System performance is another critical factor that determines the success of the gaming console. The console must be able to run multiple gaming programs simultaneously without causing lag or other performance issues. The console should also have sufficient memory and processing power to handle the demands of modern games. To ensure optimal performance, the gaming console should be tested thoroughly under different conditions, such as running multiple games at once or running graphics-intensive games.

Finally, the compatibility of the gaming programs is an essential consideration. The gaming console must be able to access and run multiple gaming programs written in C++. This requires careful attention to the file format, libraries, and dependencies used by each game. Additionally, the console should be able to handle different input devices, such as keyboards, mice, and game controllers, to ensure that users can play games using their preferred input method.

In conclusion, the design and implementation of a gaming console that can access multiple gaming programs in C++ requires careful consideration of several factors. These include the programming language used, the user interface design, the system performance, and the compatibility of the gaming programs. With proper planning and attention to detail, it is possible to create a gaming console that provides a seamless and immersive gaming experience to users. The success of the gaming console ultimately depends on its ability to meet the needs and expectations of its users, which can be achieved by careful testing and user feedback.

REFERENCE

1. https://app.diagrams.net/
2. https://www.canva.com/
3. https://www.lucidchart.com/blog/uml-diagram-templates
4. https://google.com
5. https://www.wikipedia.org/
6. https://stackoverflow.com/
7. https://www.geeksforgeeks.org/