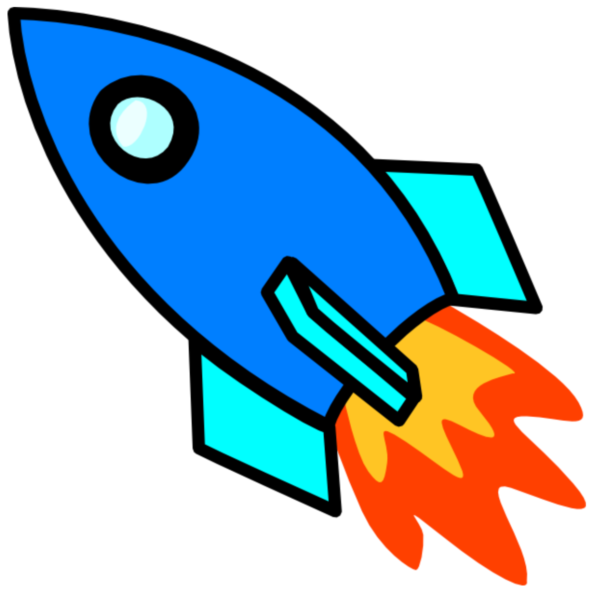


rigel

A cross-platform space adventure videogame

Arnav Dhamija

Grade: **12 B**
Academic year: **2014-15**
National Public School HSR Layout, Bangalore

powered by

SFML

ACKNOWLEDGEMENTS

I would like to thank Nalina ma'am for teaching me the underlying concepts to make this project possible. Moreover, I would like to thank my father, Vikas Dhamija for inspiring me to learn and implement a high-level graphics library, SFML, for this game. Moreover, online resources such as StackExchange and sfml-dev.org were very useful for fixing bugs and for learning new C++ concepts.

INDEX

OVERVIEW OF C++

EVOLUTION OF C++

The C++ programming language has a history going back to 1979, when Bjorne Stroustrup was doing work for his Ph.D. thesis. One of the languages Stroustrup had the opportunity to work with was a language called Simula, which as the name implies is a language primarily designed for simulations. Shortly thereafter, he began work on "“C with Classes”", which as the name implies was meant to be a superset of the C language. His goal was to add object-oriented programming into the C language, which was and still is a language well-respected for its portability without sacrificing speed or low-level functionality. His language included classes, basic inheritance, inlining, default function arguments, and strong type checking in addition to all the features of the C language.

The first “C with Classes” compiler was called Cfront, which was derived from a C compiler called CPre. It was a program designed to translate “C with Classes” code to ordinary C. A rather interesting point worth noting is that Cfront was written mostly in “C with Classes”, making it a self-hosting compiler (a compiler that can compile itself). Cfront would later be abandoned in 1993 after it became difficult to integrate new features into it, namely C++ exceptions.

In 1983, the name of the language was changed from “C with Classes” to C++. The ++ operator in the C language is an operator for incrementing a variable, which gives some insight into how Stroustrup regarded the language.

FEATURES

C++ is the multi paradigm, compiled, free form, general purpose, statistically typed programming language. This is known as middle level language as it comprises of low level and high level language features.

The first commercial implementation of the C++ was released in 1985 and before that the name of language was changed to “C++”. And some new features were added to the language and the main features of the C++ are:

1. Classes
2. Inheritance
3. Data abstraction and encapsulation
4. Polymorphism
5. Dynamic Binding
6. Message passing

ADVANTAGES

1. **Portable**: the C++ standard is consistent
2. **Industrial**: evolved to satisfy the needs of software engineers, not computer scientists
3. **Efficient**: Compiles into highly optimized CPU-specific machine code with little or no runtime overhead.
4. **Multi-paradigm**: allows the use and penalty-free mixing of procedural, OOP, generic programming, functional programming, etc.
5. **Strictly statically typed** (unlike Python for example): a large amount of logic and calculations can be performed at compile time (by the type checking/inferring system)
6. Has **deterministic memory management** (as opposed to Java, C#, and other languages with garbage collectors): the life time of every object is known with absolute precision, which makes destructors useful and RAII possible.

INTRODUTION

PROJECT SUMMARY

Rigel is the name of the brightest star in the Orion constellation with a mass of 21 solar masses and a surface temperature of 12100 K. This project is named after this star because of its linkage to space.

In this game, a flying craft (an Airplane, a Twitter Bird, or a Helium Balloon) must dodge several obstacles in their path to make it to the Helix Nebula to attain their goal of travelling through the universe. On the way, one might encounter powerups which can increase the rate of increase of the score.

The game is divided into 10 levels in successively increasing difficulty and length. Each level has its own unique background and theme.

TECHNICAL DETAILS

The game uses the Simple and Fast Media Layer (SFML) 2.1 graphics library to implement graphics off the C++ code. SFML implements graphics via GPU hardware acceleration using OpenGL methods.

File manipulation is highly involved in this game with each level requiring four files, leading to a total of 42 files used in the entire game. To reduce file size, the ios::binary flag has been used where required. The game has been designed in an object oriented manner with classes, structures, and linked lists.

The game has been designed such that no system specific library is used to ensure portability across Linux, Windows, and Mac OS X on the latest versions of the gcc compiler.

Furthermore, the C++ file is compiled with the gcc standard –O2 flag to optimize the code to provide the best performance.

A separate file called **`levelRNG.cpp`** automatically generates all the level files using the C++ standard pseudo RNG.

SYSTEM REQUIREMENTS

HARDWARE

To run Rigel, the target computer must have at least:

- CPU: Intel Pentium Dual-Core CPU or AMD-equivalent
- RAM: 1 GB
- Graphics card: Most modern CPU integrated graphics cards should be sufficient to run this game. However, using Intel IvyBridge or newer integrated graphics cards are recommended. Dedicated graphics cards may offer better performance
- Space on Hard Disk: 30 MB

SOFTWARE

- Microsoft Windows: Using Windows XP or newer is sufficient.
- Linux: Using the latest version of any Linux distribution is sufficient.

SFML must be installed as a pre-requisite to using this game. More details on installing SFML can be found in the Installation heading of this document. Both 32-bit and 64-bit versions Windows and Linux are supported.

INSTALLATION (MICROSOFT WINDOWS)

First, you must unpack the SFML version needed for your computer from the folder called SFML in the project CD. The required files can be downloaded from the SFML website.

There are multiple variants of gcc for Windows, which are incompatible with each other (different exception management, threading model, etc.). Make sure that you pick up the right package according to the version that you use. If you don't know, check which of the libgcc_s_sjlj-1.dll or libgcc_s_dw2-1.dll file you have in your MinGW/bin folder. If you're using the version of MinGW shipped with Code::Blocks, you probably have a SJLJ version.

Copying headers and libraries to your installation of MinGW is not recommended, it's better to keep libraries in their own separate location, especially if you intend to use several versions of the same library, or several compilers.

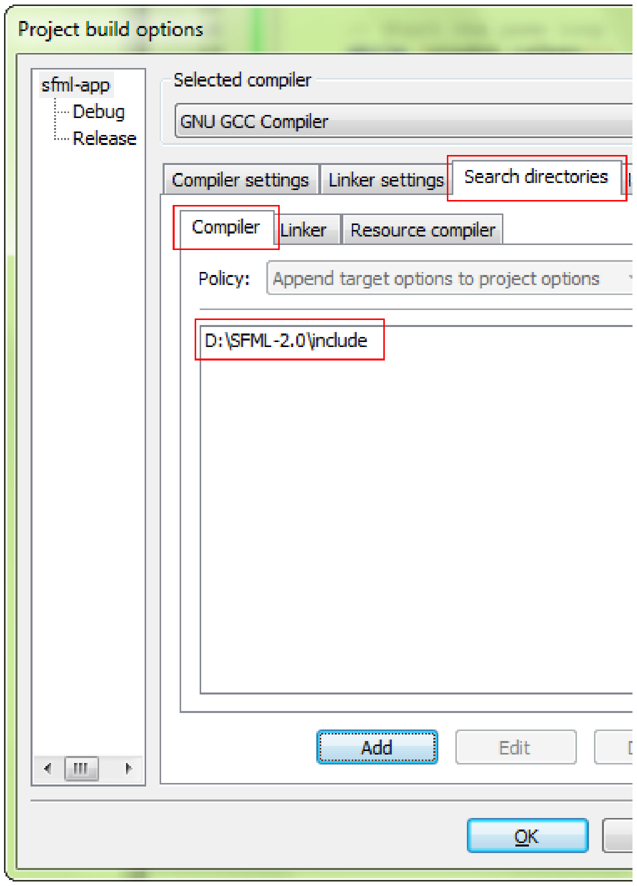
Selecting the Global Compiler Settings from the Settings tab in Code::Blocks, follow the following instructions to prepare Code::Blocks for SFML projects.

Now we need to tell the compiler where to find the SFML headers (.hpp files), and the linker where to find the SFML libraries (.a files).

In the project's "Build options", "Search directories" tab, add:

- The path to the SFML headers (*<sfml-install-path>/include*) to the Compiler search directories
- The path to the SFML libraries (*<sfml-install-path>/lib*) to the Linker search directories

These paths are the same in both Debug and Release configuration, so you can set them globally for your project.


Project build options

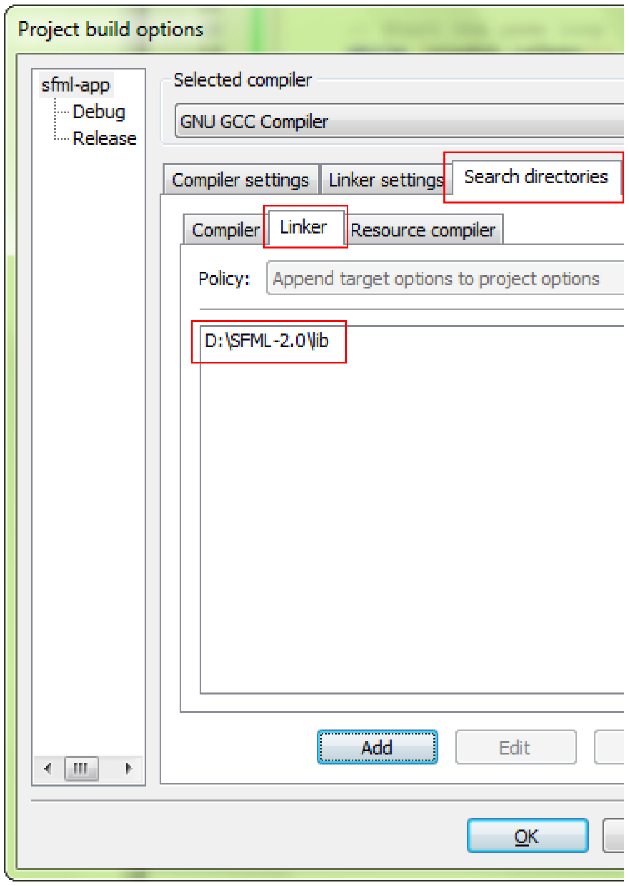
sfml-app
Debug
Release

Selected compiler: GNU GCC Compiler

Search directories > Compiler

Policy: Append target options to project options

D:\SFML-2.0\include


Project build options

sfml-app
Debug
Release

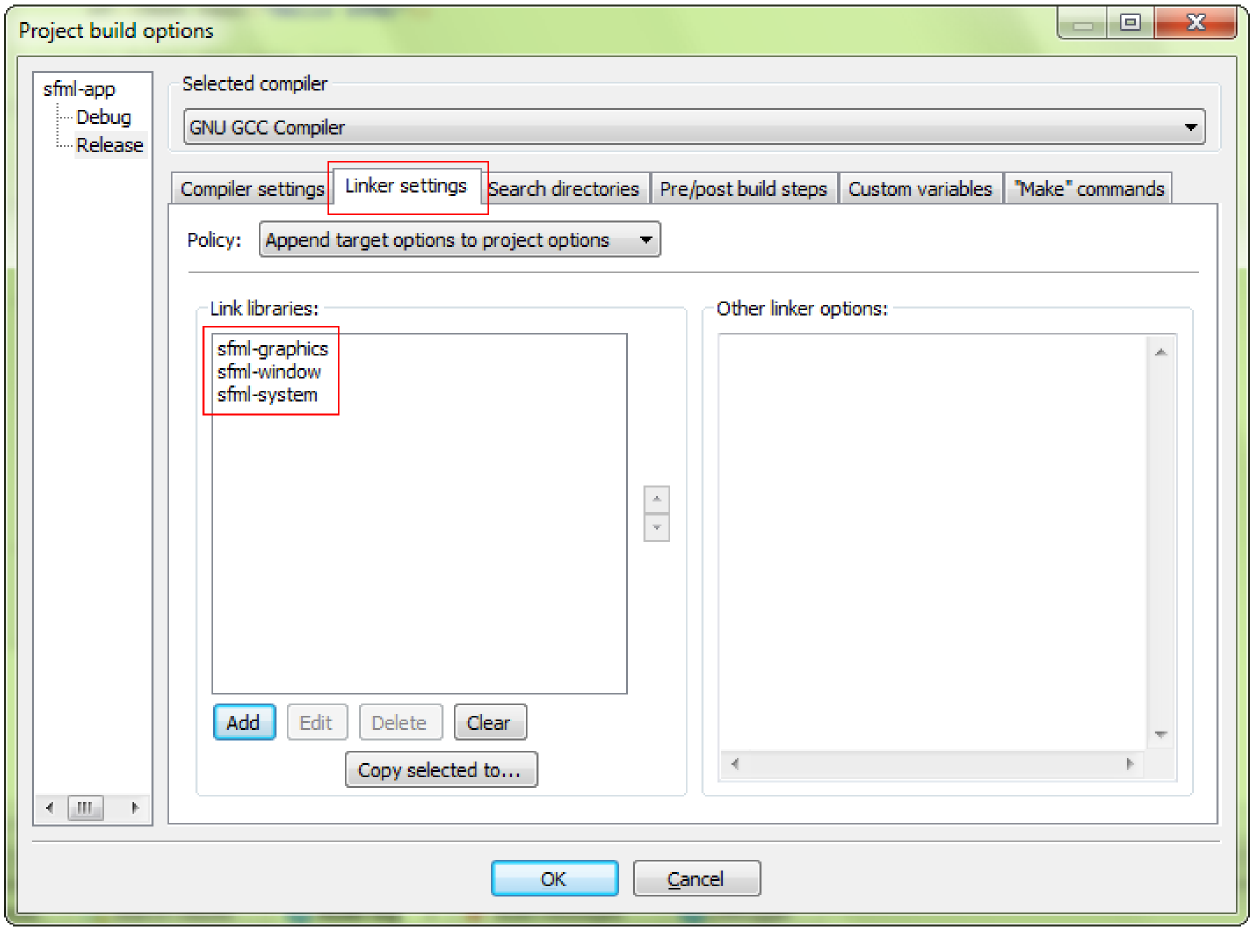
Selected compiler: GNU GCC Compiler

Search directories > Linker

Policy: Append target options to project options

D:\SFML-2.0\lib

The next step is to link your application to the SFML libraries (.a files) that your code will need. SFML is made of 5 modules (system, window, graphics, network and audio), and there's one library for each of them.
Libraries must be added in the project's properties, "Linker settings" tab, "Link libraries" list. Add all the SFML libraries that you need, for example "sfml-graphics", "sfml-window" and "sfml-system" (the "lib" prefix and the ".a" extension must be omitted).



Selected compiler: GNU GCC Compiler

Policy: Append target options to project options

Link libraries:

sfml-graphics
sfml-window
sfml-system

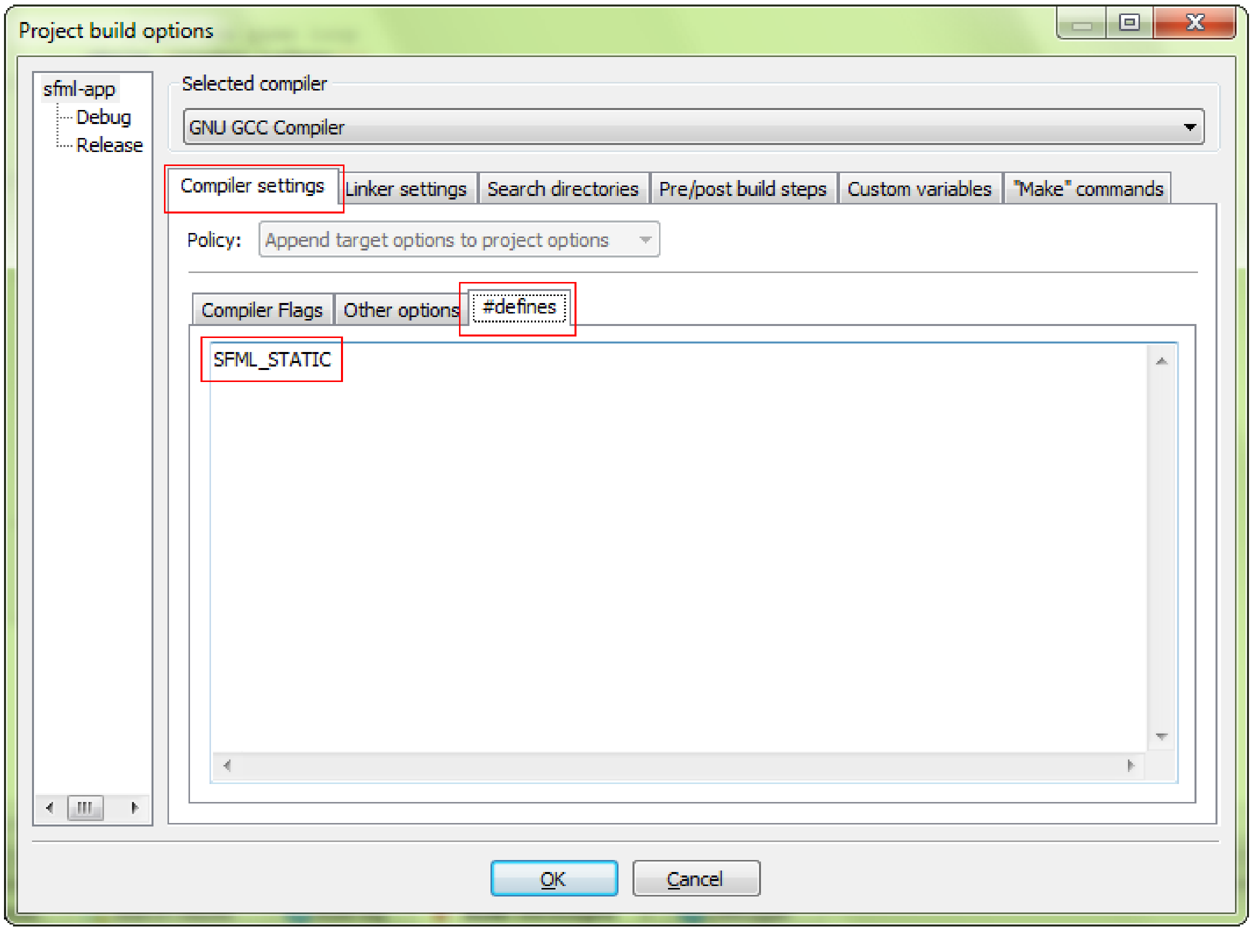
Other linker options:

It is important to link to the libraries that match the configuration: "sfml-xxx-d" for Debug, and "sfml-xxx" for Release. A bad mix may result in crashes.

When linking to multiple SFML libraries, make sure that you link them in the right order, it is very important for gcc. The rule is that libraries that depend on other libraries must be put first in the list. Every SFML library depends on sfml-system, and sfml-graphics also depends on sfml-window. So, the correct order for these three libraries would be: sfml-graphics, sfml-window, sfml-system -- as shown in the screen capture above.

The settings shown here will result in your application being linked to the dynamic version of SFML, the one that needs the DLL files. If you want to get rid of these DLLs and have SFML directly integrated to your executable, you must link to the static version. Static SFML libraries have the "-s" suffix: "sfml-xxx-s-d" for Debug, and "sfml-xxx-s" for Release.

In this case, you'll also need to define the SFML_STATIC macro in the preprocessor options of your project.



Selected compiler: GNU GCC Compiler

Compiler settings

Policy: Append target options to project options

#defines

```
SFML_STATIC
```

After completing the previous steps, copy all the .dll files in the SFML bin folder to the same folder as where the rigel C++ file is. Now, simply Build and Run the program from Code::Blocks.

INSTALLATION (LINUX)

In this case, setting up SFML on Ubuntu is taken as an example.

Firstly, install the needed SFML libraries by entering the following command in the Terminal:

```
sudo apt-get install libsfml-dev
```

Then, after changing the working directory to where the Rigel C++ file is located, build it from the Terminal with the command:

```
g++ -c rigel.cpp
```

```
g++ main.o rigel –o –O2  -lsfml-graphics -lsfml-window -lsfml-system
```

Finally run it with the command

```
./rigel
```

HEADER FILES

The following functions and objects have been used in this project from their respective header files:

`iostream`

- `cin`
- `cout`

`fstream`

- `ifstream`
- `ofstream`
- `ios`
- `<fstream-object>.open()`
- `<fstream-object>.close()`
- `<fstream-object>.good()`

`SFML/Graphics.hpp`

- `sf::Color`
- `sf::RectangleShape`
- `sf::CircleShape`
- `sf::RenderWindow`
- `sf::Vector2f`
- `sf::Texture`
- `sf::Font`
- `sf::Text`
- `<sfml-object>.loadFromFile(…)`
- `<sfml-object>.getGlobalBounds()`
- `<sfml-object>.set<Value>(…)`
- `<sfml-object>.clear()`
- `<sfml-object>.setPosition(…)`

`sstream`

- `ostringstream`

`string`

- `string`

CLASSES AND OBJECTS

`class objectAttributes`
Class containing structures to be inherited by other classes

`struct rectangleData`
Specifies how rectangles for obstacles are visually designed

`struct rectangleColumn`
Consists of two rectangle data objects to create a column of rectangles

`struct powerupData`
Specifies how powerups are visually designed

`class blocks`
Specifies setting values and reading all the blocks from a file and animates the transition of all the blocks through the screen

`class powerups`
Specifies checking setting values and reading all the powerups from a file and animates the transition of all the powerups through the screen

`class fly`
Specifies the object the player can control on the screen with inbuilt collisions and attributes

`struct node`
Nodes required for the linked list of scores

`class scoresList`
Reads scores from a file and stores it in the linked list

FUNCTIONS

BLOCKS CLASS

`blocks()`
Constructor for blocks objects

`~blocks()`
Destructor for blocks objects

`static void drawBlocks(blocks *blockList, sf::RenderWindow &window)`
Draws blocks on the screen with SFML methods

`static void fileRead(blocks *blockList)`
Reads blocks from each file for the levels

`static void getBlockCount()`
Gets the number of blocks from the specified file

`static void moveBlocks(blocks *blockList)`
Animates all the blocks by moving them with each frame by changing the position vector of the blocks

`void setValues(int x)`
Set colours, dimensions, and placements for blocks

POWERUPS CLASS

`powerups()`
Constructor for powerups objects

`~powerups()`
Destructor for powerups objects

`void drawPowerup(sf::RenderWindow &window)`
Draws powerups on the screen using SFML methods

`static void fileReadPowerups(powerups *powerupsList)`
Reads powerups from each file for the levels

`static void getPowerupCount()`
Gets the number of powerups from the specified file

`void movePowerup()`
Animates all the blocks by moving them with each frame by changing the position vector of the powerups

`void setValues()`
Set colours, dimensions, and placements for powerups

FLY CLASS

`void collisionCheck(blocks *blockList)`
Overloaded function checking whether flying craft strikes a block triggering a crash

`void collisionCheck(powerups *powerupsList, bool &doubleScore)`
Overloaded function checking whether flying craft strikes a powerup triggering a Double Score condition

`fly(int a, int b, const char *picture, blocks *blockList, int type)`
Overloaded parameterized constructor for fly objects

`fly()`
Overloaded unparameterized constructor for fly objects

`~fly()`
Destructor for fly objects

`int flyType()`
Returns the type of flying craft

`void jump(float dt)`
Triggers the jump function, the flag specifies which way to animate the object

`void movement(float dt, int flag)`
Animates the flying object by changing its position vector

`void setValues(int a, int b, const char *picture, blocks *blockList, int value)`
Sets values for the fly object

`static int typeNumber(string imageFile)`
Sets the type of flying craft based on the image sent to the function

SCORESLIST CLASS

`node *createNode(int x)`
Creates nodes for the linked list of scores

`void displayAllNodes(node *newNode)`
Traverses the linked list using pointers and display it to a terminal screen

`void getNodes()`
Reads all the nodes from the specified file

`void insertNode(node *newNode)`
Inserts the nodes ready to be taken into the linked list

`scoresList()`
Constructor for scoresList object

`~scoresList()`
Destructor for scoresList object

`void writeScores(int score)`
Writes all the scores to the file

UNCLASSIFIED FUNCTIONS

`int checkLevelComplete(int distanceCovered, bool &gamePause)`
Checks whether the flying craft has crossed the total distance to be covered in the entire level

`void deleteScores()`
Deletes all scores from the file by truncating the file

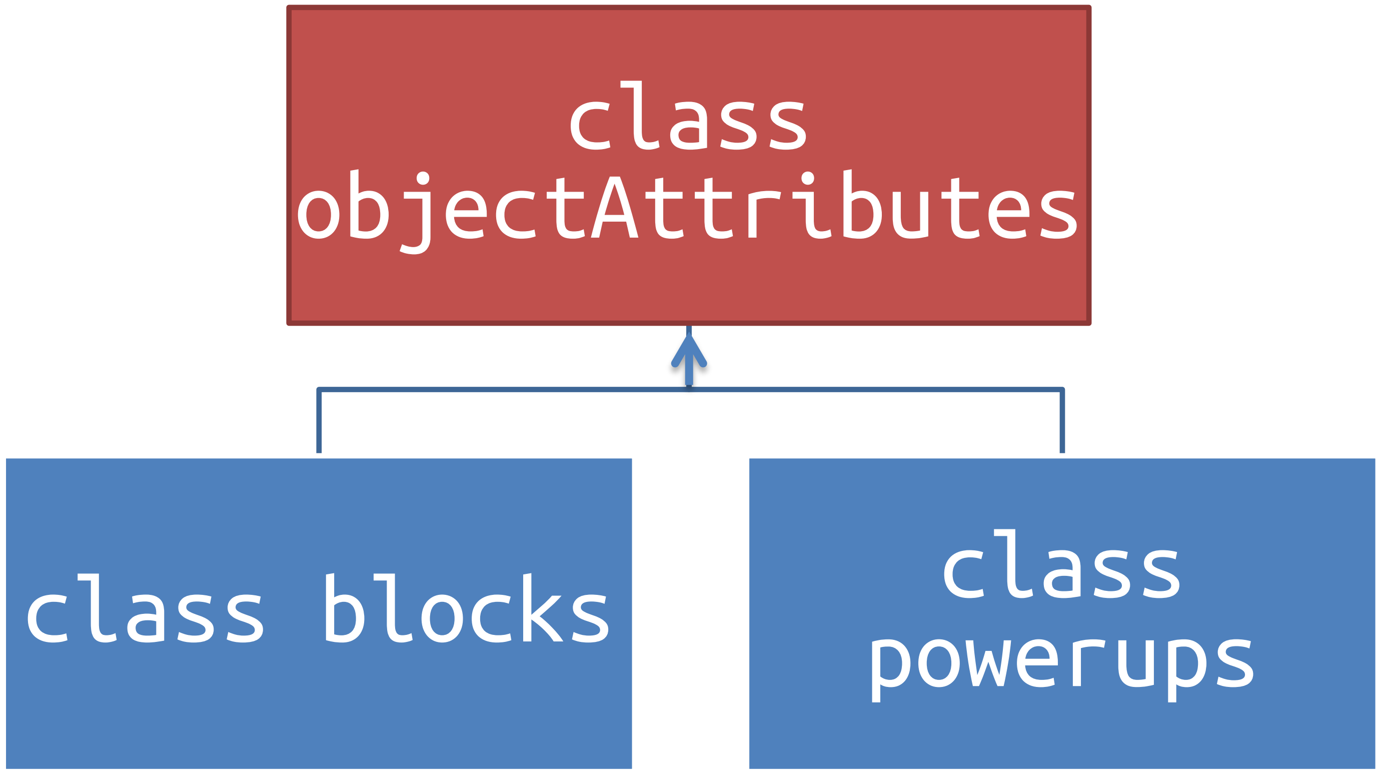
`void gameTitle()`
Displays the game title in stylized ASCII font on the terminal screen

`int menu(bool &menuCheck, string &imageFile, scoresList listOfScores)`
Prints the menu to the screen and returns whether the menu has finished processing inputs

`string to_string(T value)`
Takes template argument to convert a float value to a string to be returned for other functions to use

`bool checkFiles()`
Checks whether all required files are present in the /data folder and terminates the program if any files are missing

DATA FLOW DIAGRAM



SOURCE CODE

RIGEL SOURCE CODE

This is the source code for the game. Program logic is handled completely with C++ and graphics and window creation is handled with SFML. Levels and powerup placement is read from files generated from the levelRNG file.

```
/*
          "                  ""#
  m mm  mmm     mmmm   mmm     #
  #"  "   #    #" "#  #"  #    #
  #       #    #   #  #""""    #
  #     mm#mm  "#m"#  "#mm"    "mm
                m  #
                 ""

 * rigel: A cross-platform space adventure video-game
 * Author: Arnav Dhamija, Grade XII-B, 2014-15, NPS-HSR, Bangalore
 * Programmed using SFML 2.1 and C++
 * Please refer to the project documentation for instructions on installing SFML
 */
#include <iostream>
#include <fstream>
#include <SFML/Graphics.hpp>
#include <sstream>
#include <string>

using namespace std;

//float to string converter
template <typename T>
string to_string(T value)
{
    ostringstream oss;
    oss << value;
    return oss.str();
}

const int numberOfLevels = 9;
int powerupCount;
int blockCount;
float gravity = 400.f;

bool crashed = false;
bool levelComplete = false;
int distanceCovered;
int speedValue = 25;

int levelNum;
string level = "data/level";
string levelNumber;
string backgroundFile;

class objectAttirbutes //class defined to be inherited by other classes
{
	protected:
		struct rectangleData
		{
			int height;
			int width;
			int posY;
			int posX;
		};
		struct rectangleColumn
		{
			rectangleData top;
			rectangleData bottom;
		};
		struct powerupData
		{
			int radius;
			int posY;
			int posX;
			int type;
		};
		struct shape
		{
			sf::RectangleShape shapeTop;
			sf::RectangleShape shapeBottom;
		};
};

class blocks:private objectAttirbutes
{
	private:
	public:
		rectangleColumn value;
		static int offset;
		blocks()
		{
		}
		~blocks()
		{
		}
		shape drawThis;
		static void getBlockCount()
		{
			string file = level + levelNumber + ".data";
			ifstream input(file.c_str(), ios::binary);
			input >> blockCount;
			input.close();
		}
		static void fileRead(blocks *blockList)
		{
			int i;
			string file = level + levelNumber + ".map";
			ifstream input(file.c_str(), ios::binary);
			for(i = 0; i < blockCount; i++)
			{
				input.read((char*)&blockList[i].value, sizeof(blockList[i].value));
			}
			input.close();
		}
		void setValues(int x)
		{
			if(levelNumber == "0" || levelNumber == "1" || levelNumber == "3" ||
levelNumber == "4" || levelNumber == "9") //set for block colors
			{
				drawThis.shapeTop.setFillColor(sf::Color(128, 0, 255));
				drawThis.shapeBottom.setFillColor(sf::Color(128, 0, 255));
				drawThis.shapeTop.setOutlineColor(sf::Color::Black);
				drawThis.shapeBottom.setOutlineColor(sf::Color::Black);
			}
			else if(levelNumber == "2")
			{
				drawThis.shapeTop.setFillColor(sf::Color(255, 0, 30));
				drawThis.shapeBottom.setFillColor(sf::Color(255, 0, 30));
				drawThis.shapeTop.setOutlineColor(sf::Color::Black);
				drawThis.shapeBottom.setOutlineColor(sf::Color::Black);
			}
			else
			{
				drawThis.shapeTop.setFillColor(sf::Color(30, 150, 30));
				drawThis.shapeBottom.setFillColor(sf::Color(30, 150, 30));
				drawThis.shapeTop.setOutlineColor(sf::Color(30, 30, 30));
				drawThis.shapeBottom.setOutlineColor(sf::Color(30, 30, 30));
			}

			drawThis.shapeTop.setOutlineThickness(-7.5);
			drawThis.shapeBottom.setOutlineThickness(-7.5);
			value.top.posX = x;
			value.bottom.posX = x;
			drawThis.shapeTop.setSize(sf::Vector2f(value.top.width, value.top.height));
			drawThis.shapeBottom.setSize(sf::Vector2f(value.bottom.width,
value.bottom.height));
			drawThis.shapeTop.setPosition(value.top.posX, value.top.posY);
			drawThis.shapeBottom.setPosition(value.bottom.posX, value.bottom.posY);
		}
		static void moveBlocks(blocks *blockList)
		{
			offset -= speedValue;
			for(int i = 0; i < blockCount; i++)
			{
				blockList[i].setValues(i * blockList[i].value.top.width + offset);
			}
			distanceCovered += speedValue;
		}
		static void drawBlocks(blocks *blockList, sf::RenderWindow &window)
		{
			int i;
			for(i = 0; i < blockCount; i++)
			{
				window.draw(blockList[i].drawThis.shapeTop);
				window.draw(blockList[i].drawThis.shapeBottom);
			}
		}
};

int blocks::offset = 0; //initialises position for where blocks will start moving

class powerups:private objectAttirbutes
{
	private:
	public:
		int type;
		sf::Texture design;
```

```
            powerupData value;
            sf::CircleShape body;
            powerups()
            {
            }
            ~powerups()
            {
            }
            static void fileReadPowerups(powerups *powerupsList)
            {
                  string file = level + levelNumber + ".pow";
                  ifstream input(file.c_str(), ios::binary);
                  int i;
                  for(i = 0; i < powerupCount; i++)
                  {
              input.read((char*)&powerupsList[i].value, sizeof(powerupsList[i].value));
                  }
                  input.close();
            }
            void setValues()
            {
                  body.setRadius(value.radius);
                  body.setPosition(value.posX, value.posY);
                  body.setFillColor(sf::Color::Blue);
            }
            void drawPowerup(sf::RenderWindow &window)
            {
                  window.draw(body);
            }
            static void getPowerupCount()
            {
                  string file = level + levelNumber + ".data";
                  ifstream input(file.c_str(), ios::binary);
           input >> powerupCount;
           input >> powerupCount;
           input.close();
       }
       void movePowerup()
            {
                  value.posX -= speedValue;
                  body.setPosition(value.posX, value.posY);
            }
};

class fly
{
      private:
            struct posVector
            {
                  int x;
                  int y;
            };
            struct velocityVector
            {
                  float y;
            };
            velocityVector velocity;
            posVector position;
            sf::Texture design;
            int type;
      public:
```

```
		sf::Sprite body;
		fly()
		{
			position.x = 0;
			position.y = 0;
		}
		fly(int a, int b, const char *picture, blocks *blockList, int type)
		{
			position.x = a;
			position.y = (blockList[1].value.top.posY + blockList[1].value.bottom.posY) /
2;
			if(!design.loadFromFile(picture));
			if(type == 0) //for speedVal of Airplane
			{
				speedValue = 25;
			}
			else if(type == 1) //for speedVal of Twitter Bird
			{
				speedValue = 15;
			}
			else if(type == 2) //for setting values of Helium Balloon
			{
				speedValue = 10;
				gravity = -200.0f; //controls are inverted!
			}
			body.setTexture(design);
			body.setColor(sf::Color(255, 255, 255, 255));
		}
		~fly()
		{
		}
		void setValues(int a, int b, const char *picture, blocks *blockList, int value)
		{
			position.x = a;
			position.y = (blockList[1].value.top.posY + blockList[1].value.bottom.posY) /
2;
			if(!design.loadFromFile(picture))
			{
				cout << "\nI/O Error";
			}
			body.setTexture(design);
			type = value;
			if(type == 0)
			{
				speedValue = 25;
			}
			else if(type == 1)
			{
				speedValue = 15;
			}
			else if(type == 2)
			{
				speedValue = 10;
				gravity = -200.0f;
			}
			body.setColor(sf::Color(255, 255, 255, 255));
		}
		void movement(float dt, int flag)
		{
			if(type == 0 || type == 1)
			{
				if(flag == 1)
				{
					velocity.y -= 600.f;
				}
				if(velocity.y < gravity)
				{
					velocity.y += 10.f;
				}
				else if(velocity.y > gravity)
				{
					velocity.y = gravity;
				}
			}
			if(type == 2) //for helium balloon
			{
				if(flag == 1)
				{
					velocity.y += 500.f;
				}
				if(velocity.y < gravity)
				{
					velocity.y = gravity;
				}
				else if(velocity.y > gravity)
				{
					velocity.y -= 10.f;
				}
			}
			position.y += velocity.y * dt;
			body.setPosition(position.x, position.y);
		}
		void jump(float dt)
		{
			movement(dt, 1);
		}
		void collisionCheck(blocks *blockList)
		{
			for(int i = 0; i < blockCount; i++)
			{

	if(body.getGlobalBounds().intersects(blockList[i].drawThis.shapeTop.getGlobalBounds()) ||
body.getGlobalBounds().intersects(blockList[i].drawThis.shapeBottom.getGlobalBounds()))
//condition checks whether flying object collides with bottom or top block
				{
					if(!design.loadFromFile("sprites/explosion.png"));
					body.setTexture(design);
					crashed = true;
				}
			}
		}
		void collisionCheck(powerups *powerupsList, bool &doubleScore)
		{
			for(int i = 0; i < powerupCount; i++)
			{

	if(body.getGlobalBounds().intersects(powerupsList[i].body.getGlobalBounds()))
				{
					if(powerupsList[i].type == 0)
					{
						doubleScore = true;
					}
				}
			}
		}
		static int typeNumber(string imageFile)
		{
			if(imageFile == "sprites/airplane.png")
			{
				return 0;
			}
			if(imageFile == "sprites/bird.png")
			{
				return 1;
			}
			if(imageFile == "sprites/balloon.png")
			{
				return 2;
			}
			return -1;
		}
		int flyType()
		{
			return type;
		}
};

struct node //for linked list
{
	int value;
	node *next;
}*top, *temp, *save;

class scoresList
{
	private:
	public:
		scoresList()
		{
		}
		~scoresList()
		{
		}
		void writeScores(int score)
		{
			ofstream output("data/scores.txt", ios::binary|ios::app);
			output.seekp(0, ios::end);
			output << score << "\n";
			output.close();
		}
		void getNodes()
		{
			ifstream input("data/scores.txt", ios::binary);
			int x;
			while(input.eof() == 0)
			{
				input >> x;
				insertNode(createNode(x));
			}
			input.close();
		}
		node *createNode(int x)
		{
```

```
            temp = new node;
            temp -> value = x;
            temp -> next = NULL;
            return temp;
        }
        void insertNode(node *newNode)
        {
            if(top == NULL)
            {
                top = newNode;
            }
            else
            {
                save = top;
                top = newNode;
                newNode -> next = save;
            }
        }
        void displayAllNodes(node *newNode)
        {
            int i = 0;
            int max = 0;
            while(newNode != NULL && i <= 10)
            {
                cout << newNode -> value << "\n";
                if(newNode -> value > max)
                {
                    max = newNode -> value;
                }
                newNode = newNode -> next;
                i++;
            }
            cout << "Highest score over last 10 plays: " << max << "\n";
        }
};

int checkLevelComplete(int distanceCovered, bool &gamePause)
{
    if(distanceCovered >= blockCount * 100)
    {
        levelComplete = true;
        gamePause = true;
        levelNumber = to_string(++levelNum);
        return 1;
    }
    return 0;
}

void gameTitle()
{
    cout << "           *                  **#" << "\n";
    cout << "    m mm  mmm     mmmm   mmm     #" << "\n";
    cout <<      "     #*  *   #    #* *#  #*  #    #" << "\n";
    cout << "    #       #    #   #  #****    #" << "\n";
    cout << "    #     mm#mm  *#m*#  *#mm*    *mm" << "\n";
    cout << "                m  #               " << "\n";
    cout << "                 **                        " << "\n";
    cout << "\nRIGEL! A cross platform videogame";
    cout << "\nProgrammed by Arnav Dhamija";
    cout << "\n--------------------------------------------\n";
}
```

```
void deleteScores()
{
	ofstream output("scores.txt", ios::trunc); //ios::trunc will overwrite file
	output.close();
}

int menu(bool &menuCheck, string &imageFile, scoresList listOfScores)
{
	int choice;
	int levelChoice;
	char option;
	cout << "\nSelect option:\n\n";
	cout << "1 - PLAY!\n";
	cout << "2 - Choose aircraft\n";
	cout << "3 - Scores\n";
	cout << "4 - Clear score history\n";
	cout << "5 - Controls\n";
	cout << "6 - Select Level\n";
	cout << "7 - About\n";
	cout << "8 - Exit\n";
	cin >> choice;
	switch(choice)
	{
		case 1:
			menuCheck = true;
			break;
		case 2:
			cout << "a - Airplane (default): The Airplane is faster than the other two and 
provides a decent multiplier.\n";
			cout << "b - Twitter Bird: For new players; the Twitter Bird is slower than an 
Airplane.\n";
			cout << "c - Helium Balloon: For experienced players; the Helium Balloon 
inverts controls!\n";
			cin >> option;
			switch(option)
			{
				case 'a':
					imageFile = "sprites/airplane";
					break;
				case 'b':
					imageFile = "sprites/bird";
					break;
				case 'c':
					imageFile = "sprites/balloon";
					break;
			}
			break;
		case 3:
			cout << "Previous 10 scores:\n";
			listOfScores.getNodes();
			listOfScores.displayAllNodes(top);
			top = NULL;
			break;
		case 4:
			deleteScores();
			cout << "Previous scores deleted.\n";
			break;
		case 5:
			cout << "Press <SPACE> to jump, P to pause.\n";
			break;
		case 6:
			cout << "Select level (0 -> 9)\n";
			cin >> levelChoice;
			if(levelChoice >= 0 && levelChoice < 10)
			{
				levelNum = levelChoice;
				levelNumber = to_string(levelChoice);
				cout << "\nLevel " << levelChoice << " selected\n";
			}
			else
			{
				cout << "No such level! Please try again!\n";
			}
			break;
		case 7:
			gameTitle();
			break;
		case 8:
			return 1;
	}
	return 0;
}

bool checkFiles() //checks if all needed files are present in the data folder
{
	string file;
	string levelString = "data/level";
	string backgroundString = "images/background";
	int i;
	ifstream input;
	for(i = 0; i < 10; i++)
	{
		file = levelString + to_string(i) + ".data";
		input.open(file.c_str());
		if(input.good() == false)
		{
			input.close();
			return false;
		}
		input.close();
		file = levelString + to_string(i) + ".map";
		input.open(file.c_str());
		if(input.good() == false)
		{
			input.close();
			return false;
		}
		input.close();
		file = levelString + to_string(i) + ".pow";
		input.open(file.c_str());
		if(input.good() == false)
		{
			input.close();
			return false;
		}
		input.close();
	}
	file = "data/font.ttf";
	input.open(file.c_str());
	if(input.good() == false)
	{
		input.close();
		return false;
	}
	return true;
}

int main()
{
	int score;
	sf::Clock frametime;
	sf::Texture img;
	sf::Sprite background;

	string scoreString;
	string image;

	scoresList listOfScores;

	bool gamePause = true;
	bool doubleScore = false;
	bool inPlay = true;
	bool menuCheck = false;
	bool writtenToFile = false;

	int doubleScoreCount = 0;
	int i;
	score = 0;
	gamePause = true;
	doubleScore = false;
	doubleScoreCount = 0;

	//SFML requires special objects for fonts, windows, sprites, colours, etc.
	sf::Event event;
	sf::Font guiFont;
	sf::Text scoreText, scoreValue;
	sf::Text levelCompleteText;
	sf::Text crashedText;
	sf::Text doubleScoreText;
	sf::Text multiplier;
	sf::Text multiplierValue;
	sf::Text gamePausedText;
	sf::Color fontColor = sf::Color::White;
	fly flyBody;
	string imageFile = "sprites/airplane";
	gameTitle();
	levelNumber = "0";
	levelNum = 0;

	if(checkFiles() == false) //terminates program with error code if files are missing
	{
		cout << "\nI/O Error. It looks like there are some missing files. Check whether all
the folders are present. Running the levelRNG file again might fix this.";
		return 1;
	}

	while(menuCheck == false)
	{
		if(menu(menuCheck, imageFile, listOfScores) == 1)
		{
			return 0;
		}
	}

	imageFile += ".png";

	blocks *blockList;
	powerups *powerupsList;
	//no more variable declarations after this line

	sf::RenderWindow window(sf::VideoMode(1280, 720), "rigel"); //creates window for SFML to 
draw in
	window.setFramerateLimit(60);

	while(inPlay == true) //main thread for all program logic
	{
		if(levelNumber == "1" || levelNumber == "3")
		{
			fontColor = sf::Color::Black; //for better contrast
		}
		else
		{
			fontColor = sf::Color::White;
		}

		backgroundFile = "images/background" + levelNumber + ".jpg";
		if(!img.loadFromFile(backgroundFile));

		background.setTexture(img);

		blocks::getBlockCount();
		blockList = new blocks[blockCount]; //dynamically allocated to avoid stack overflows
		blocks::fileRead(blockList);
		blocks::moveBlocks(blockList);

		flyBody.setValues(190, 300, imageFile.c_str(), blockList, 
fly::typeNumber(imageFile));
		guiFont.loadFromFile("data/font.ttf");

		scoreText.setFont(guiFont);
		scoreText.setCharacterSize(40);
		scoreText.setColor(fontColor);
		scoreText.setString("SCORE: ");
		scoreText.setPosition(0, 600);
		scoreValue.setFont(guiFont);
		scoreValue.setCharacterSize(40);
		scoreValue.setColor(fontColor);
		scoreValue.setPosition(150, 600);

		levelCompleteText.setFont(guiFont);
		levelCompleteText.setCharacterSize(70);
		levelCompleteText.setColor(sf::Color::White);
		levelCompleteText.setString("LEVEL COMPLETE!");
		levelCompleteText.setPosition(300, 200);

		crashedText.setFont(guiFont);
		crashedText.setCharacterSize(70);
		crashedText.setColor(sf::Color::Red);
		crashedText.setString("CRASHED!");
		crashedText.setPosition(300, 200);

		doubleScoreText.setFont(guiFont);
		doubleScoreText.setCharacterSize(40);
		doubleScoreText.setColor(sf::Color::Red);
		doubleScoreText.setString("DOUBLE SCORE!");
		doubleScoreText.setPosition(450, 600);

		multiplier.setFont(guiFont);
		multiplier.setCharacterSize(40);
		multiplier.setColor(sf::Color::White);
		multiplier.setString("MULTIPLIER: ");
		multiplier.setPosition(0, 550);

		multiplierValue.setFont(guiFont);
		multiplierValue.setCharacterSize(40);
		multiplierValue.setColor(sf::Color::White);
		multiplierValue.setString(to_string(speedValue / 10.0));
		multiplierValue.setPosition(270, 550);

		gamePausedText.setFont(guiFont);
		gamePausedText.setCharacterSize(30);
		gamePausedText.setColor(sf::Color::White);
		gamePausedText.setString("GAME PAUSED");
		gamePausedText.setPosition(270, 400);

		powerups::getPowerupCount();
		powerupsList = new powerups[powerupCount]; //dynamically allocated to avoid stack 
overflows
		powerups::fileReadPowerups(powerupsList);

		for(i = 0; i < powerupCount; i++)
		{
			powerupsList[i].setValues();
		}

		while (window.isOpen())
		{
			float dt = frametime.restart().asSeconds();
			while(window.pollEvent(event))
			{
				if(event.type == sf::Event::Closed)
				{
					window.close();
				}
				if(event.type == sf::Event::KeyPressed)
				{
					if(event.key.code == sf::Keyboard::Space && crashed == false)
					{
						flyBody.jump(dt);
					}
					if(event.key.code == sf::Keyboard::P && gamePause == false && 
crashed == false)

					{
						window.draw(gamePausedText);
						gamePause = true;
					}
					else
					{
						gamePause = false;
					}
				}
				if(event.type == sf::Event::KeyPressed && crashed == true)
				{
					return 0;
```

```
        }
}
if(crashed == true)
{
        scoreString = to_string(score);
        scoreValue.setString(scoreString);
        window.draw(crashedText);
        window.draw(scoreText);
        window.draw(scoreValue);
        window.display();
        gamePause = true;
        if(writtenToFile == false)
        {
                cout << "Score: " << score << "\n";
                listOfScores.writeScores(score);
                writtenToFile = true;
        }
}
if(checkLevelComplete(distanceCovered, gamePause) == 1)
{
        if(flyBody.flyType() == 2)
        {
                score -= 5;
                score *= 4;
        }
        score *= speedValue / 10;
        scoreString = to_string(score);
        scoreValue.setString(scoreString);
        window.clear(sf::Color::Black);
        window.draw(levelCompleteText);
        window.draw(multiplier);
        window.draw(multiplierValue);
        window.draw(scoreText);
        window.draw(scoreValue);
        window.display();
        break;
}
if(gamePause == false)
{
        flyBody.movement(dt, 0);
        window.clear(sf::Color::Black);
        flyBody.collisionCheck(blockList);
        flyBody.collisionCheck(powerupsList, doubleScore);
        window.draw(background);
        for(i = 0; i < powerupCount; i++)
        {
                powerupsList[i].movePowerup();
                powerupsList[i].drawPowerup(window);
        }
        if(crashed == false)
        {
                blocks::moveBlocks(blockList);
        }
        blocks::drawBlocks(blockList, window);
        window.draw(flyBody.body);
        if(doubleScore == true && doubleScoreCount < 30)
        {
                score += speedValue * 2;
                doubleScoreCount++;
                window.draw(doubleScoreText);
        }
```

```
                else if(doubleScoreCount == 30)
                {
                    score += speedValue;
                    doubleScore = false;
                    doubleScoreCount = 0;
                }
                else
                {
                    doubleScore = false;
                    score += speedValue;
                }
                scoreString = to_string(score); //SFML string requires string arguement
                scoreValue.setString(scoreString);
                window.draw(scoreText);
                window.draw(scoreValue);
                window.display();
            }
        }
        if(levelNum > numberOfLevels) //when all levels are finished
        {
            listOfScores.writeScores(score);
            cout << "Score: " << score << "\n";
            cout << "\nCongratulations on finishing! :)";
            return 0;
        }
        //reinitialise variables for next level
        distanceCovered = 0;
        blocks::offset = 0;
    }
    delete blockList;
    delete powerupsList;
    return 0; //returns 0
}
```

LEVELRNG SOURCE CODE

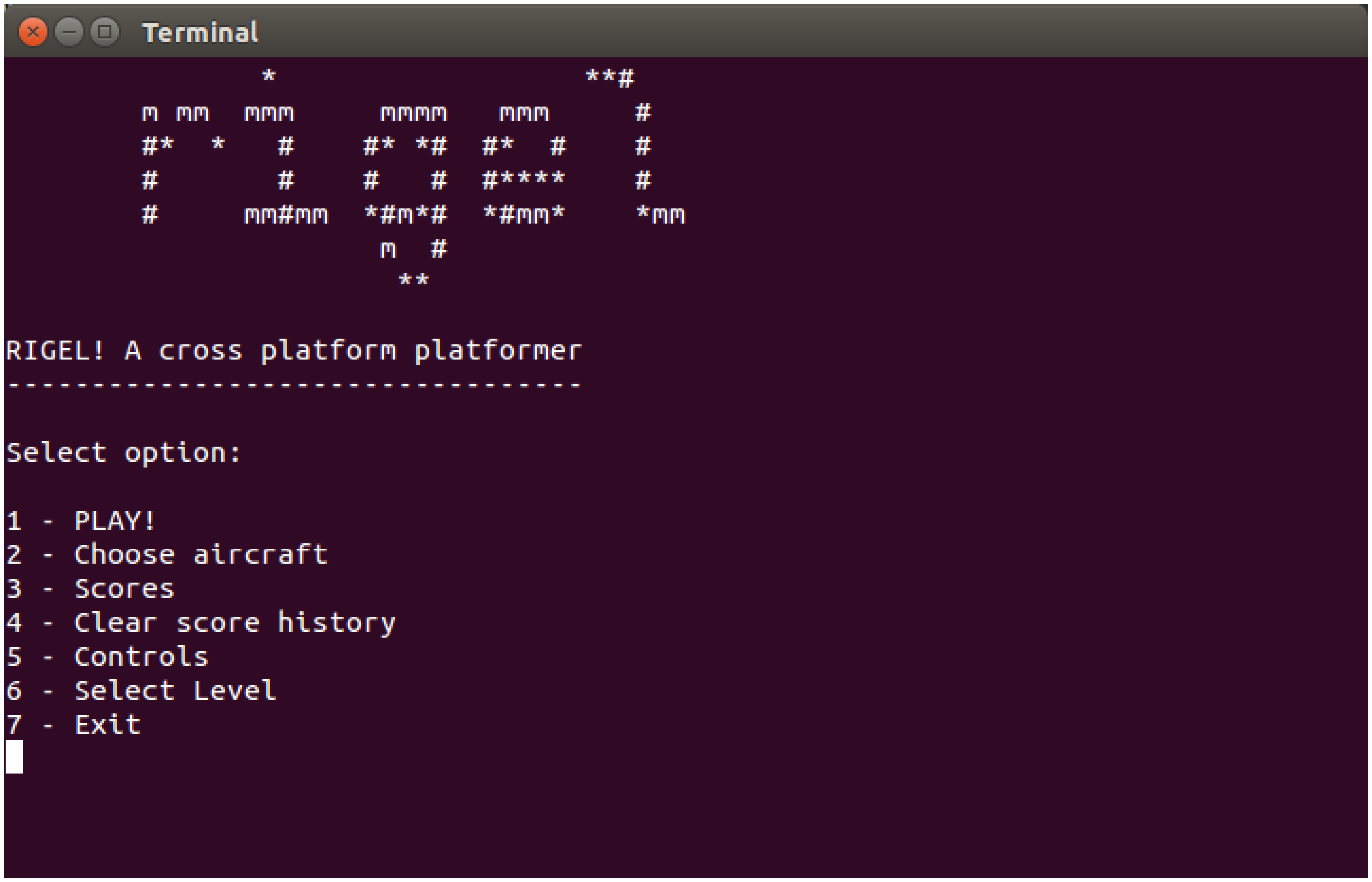
The levelRNG file automatically generates the level data for a specified number of levels. C++ `<stdlib.h>` randomization is used to randomly place the blocks on the screen.

```
/*
 * levelRNG - Helper file for rigel
 * Automatically generates all the levels in the game using C++'s pseudo-random number generator
 * Author: Arnav Dhamija, Grade XII-B, 2014-15, NPS-HSR, Bangalore
 */
#include <fstream>
#include <iostream>
#include <time.h>
#include <stdlib.h>
#include <sstream>

using namespace std;

int blockCount = 100;
int heightValue = 25;
const int widthValue = 100;

int powerupCount = 2;
int powerupRadius = 35;

string level = "data/level";
string levelNum = "0";
string background;

template <typename T>
string to_string(T value)
{
    ostringstream oss;
    oss << value;
    return oss.str();
}

void otherObjectGenerator(string levelNumber)
{
       string file = level + levelNumber + ".data";
       ofstream out2(file.c_str(), ios::binary| ios::trunc);
       out2 << blockCount << endl;
       out2 << powerupCount;
       out2.close();
}
struct powerupData
{
       int radius;
       int posY;
       int posX;
       int type;
};
struct rectangleData
{
       int height;
       int width;
       int posY;
       int posX;
};
struct rectangleColumn
```

```
{
	rectangleData top;
	rectangleData bottom;
};

void powerupGenerator(string levelNumber)
{
	powerupData lvl1;
	if(levelNumber == "0")
	{
		lvl1.type = 0;
		lvl1.posY = 360;
		lvl1.radius = powerupRadius;
		lvl1.posX = 850;
	}
	powerupData x[powerupCount];
	string file = level + levelNumber + ".pow";
	ofstream output(file.c_str(), ios::binary);
	output.write((char*)&lvl1, sizeof(lvl1));
	srand(time(NULL));
	int i;
	for(i = 0; i < powerupCount; i++)
	{
		x[i].type = 0;
		x[i].posY = 360;
		x[i].radius = powerupRadius;
		x[i].posX = rand() % 100 * blockCount;
		output.write((char*)&x[i], sizeof(x[i]));
	}
	output.close();
}
void blockGenerator(string levelNumber)
{
	string file = level + levelNumber + ".map";
	ofstream output(file.c_str(), ios::binary);
	srand(time(NULL));
	rectangleColumn x[blockCount];

	for(int i = 0; i < blockCount; i++)
	{
		x[i].top.posY = 0;
		x[i].top.width = widthValue;
		x[i].top.height = rand() % heightValue + 25;
		x[i].bottom.height = rand() % heightValue + 25;
		x[i].bottom.posY = 575 - x[i].bottom.height;
		x[i].bottom.width = widthValue;
		output.write((char*)&x[i], sizeof(x[i]));
	}
	output.close();
}
int main()
{
	string levelNumber = "0";
	float i;
	for(i = 0; i < 10; i++)
	{
		levelNumber = to_string(i);
		background = "background" + levelNumber + ".jpg";
		otherObjectGenerator(levelNumber);
		powerupGenerator(levelNumber);
		blockGenerator(levelNumber);
```

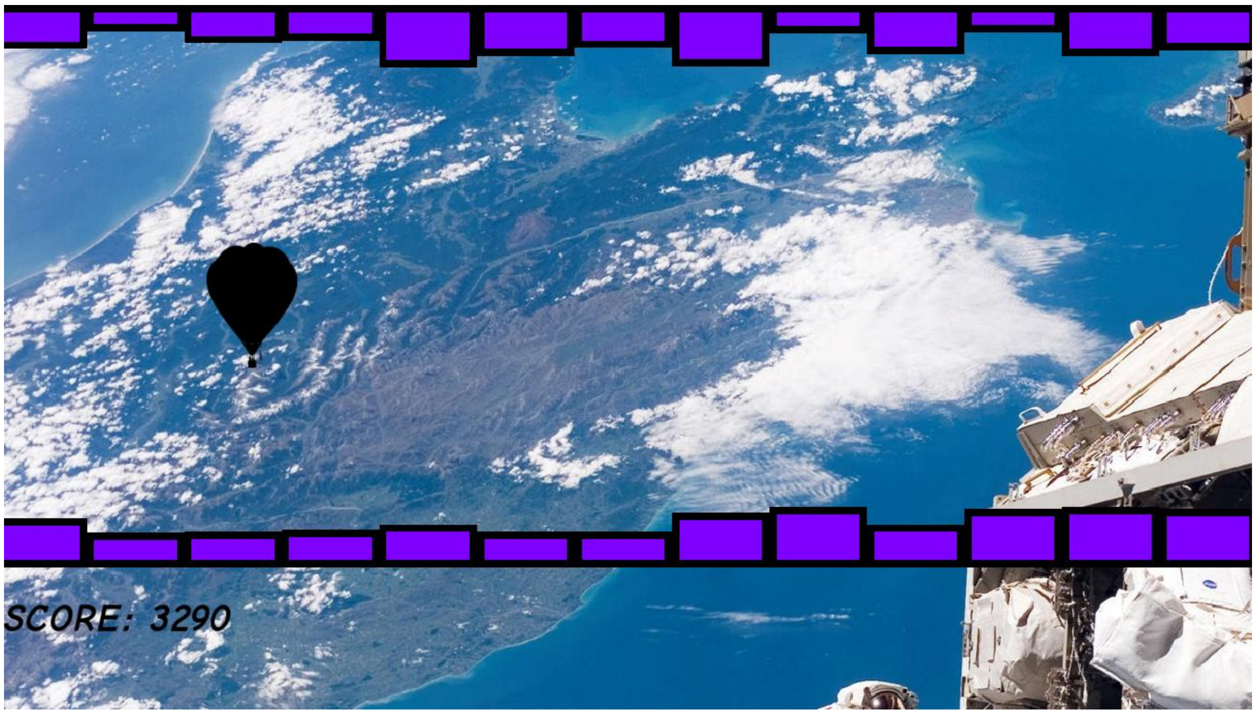
```
            if(blockCount <= 350)
            {
                  blockCount += 50;
            }
            if(heightValue <= 125)
            {
                  heightValue += 15;
            }
            powerupCount++;
      }
      cout << i * 3 << " files successfully created";
      return 0;
}
```

SCREENSHOTS

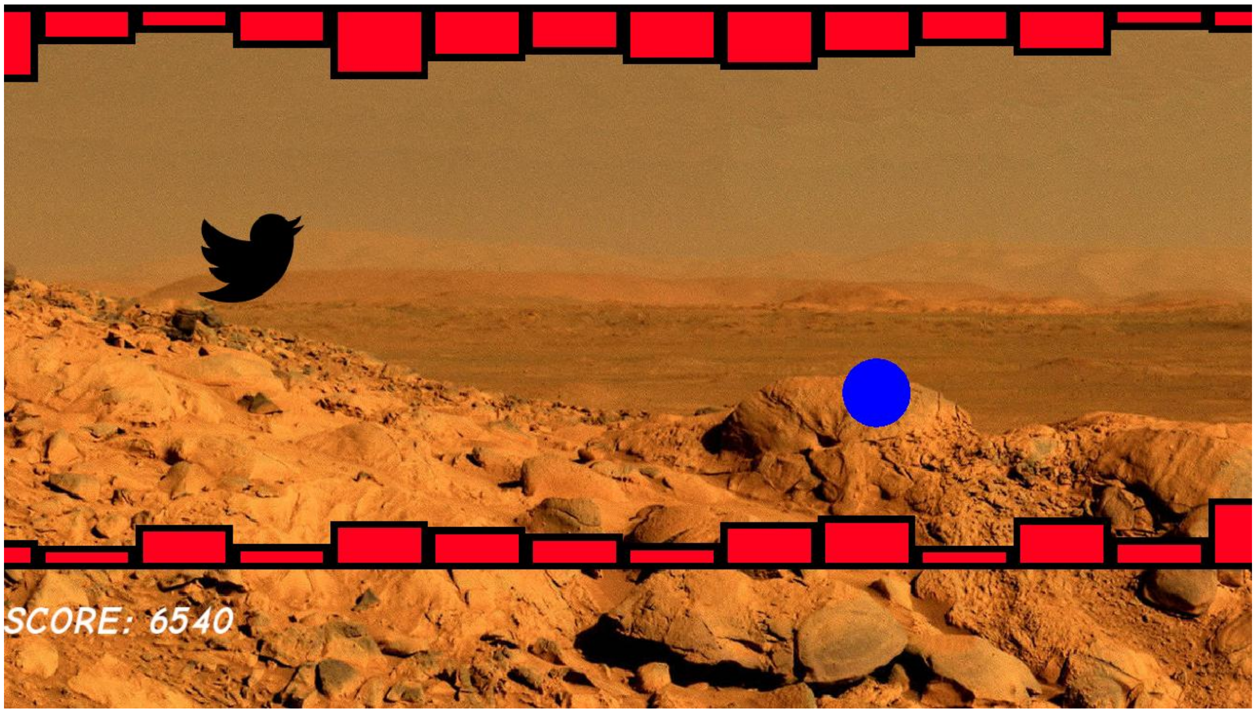


```
               *                  **#
        m mm  mmm     mmmm   mmm     #
        #*  *   #    #* *#  #*  #    #
        #       #    #   #  #****    #
        #     mm#mm  *#m*#  *#mm*    *mm
                      m  #
                       **

RIGEL! A cross platform platformer
----------------------------------

Select option:

1 - PLAY!
2 - Choose aircraft
3 - Scores
4 - Clear score history
5 - Controls
6 - Select Level
7 - Exit
```

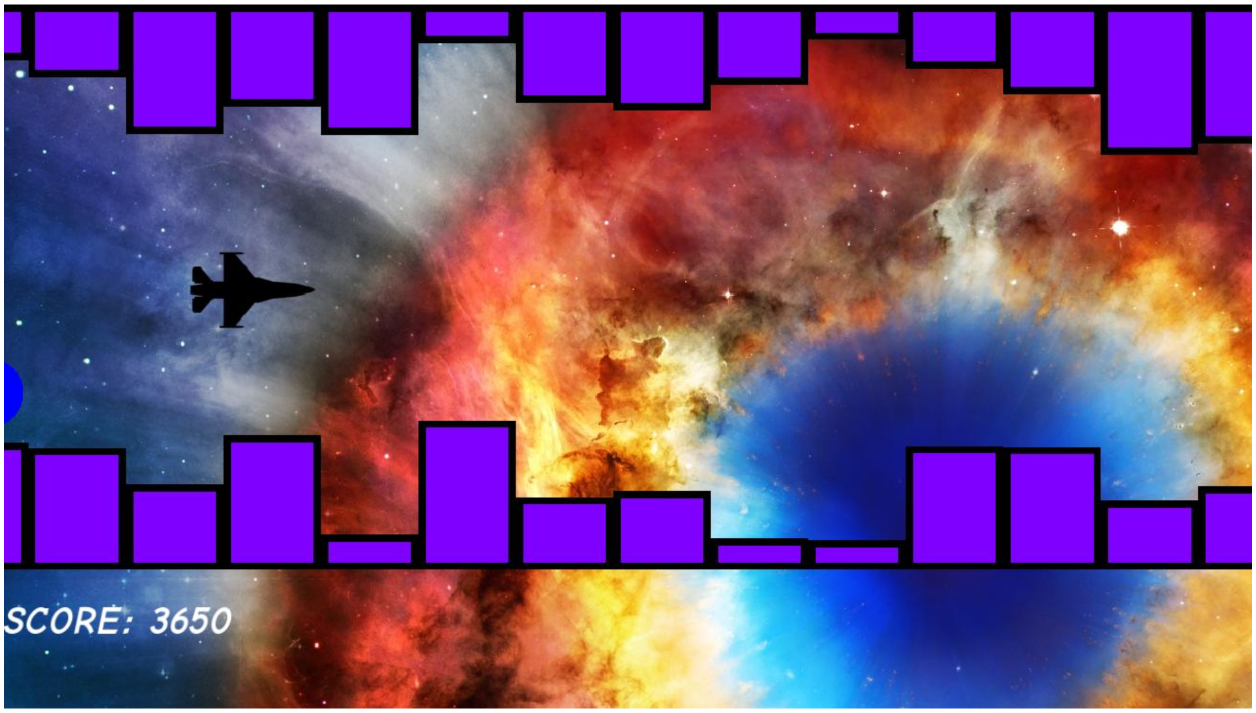
Startup menu for Rigel



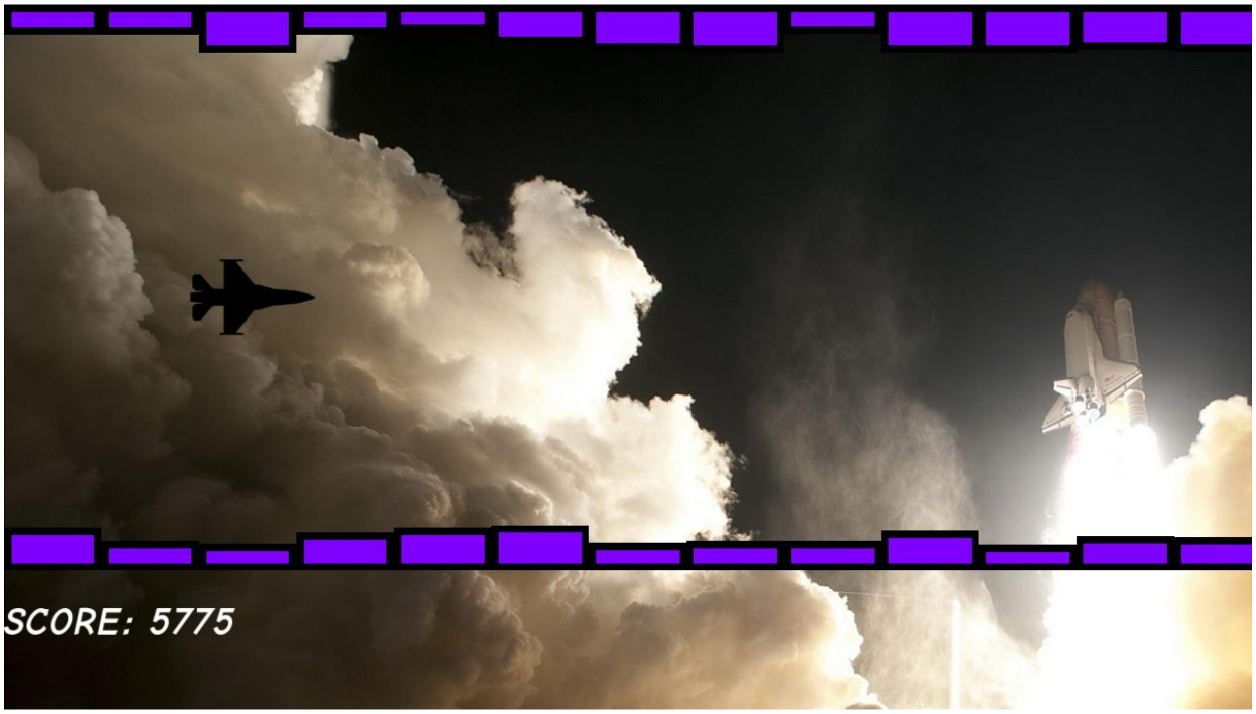
Demonstrating the Helium Balloon mode of the game



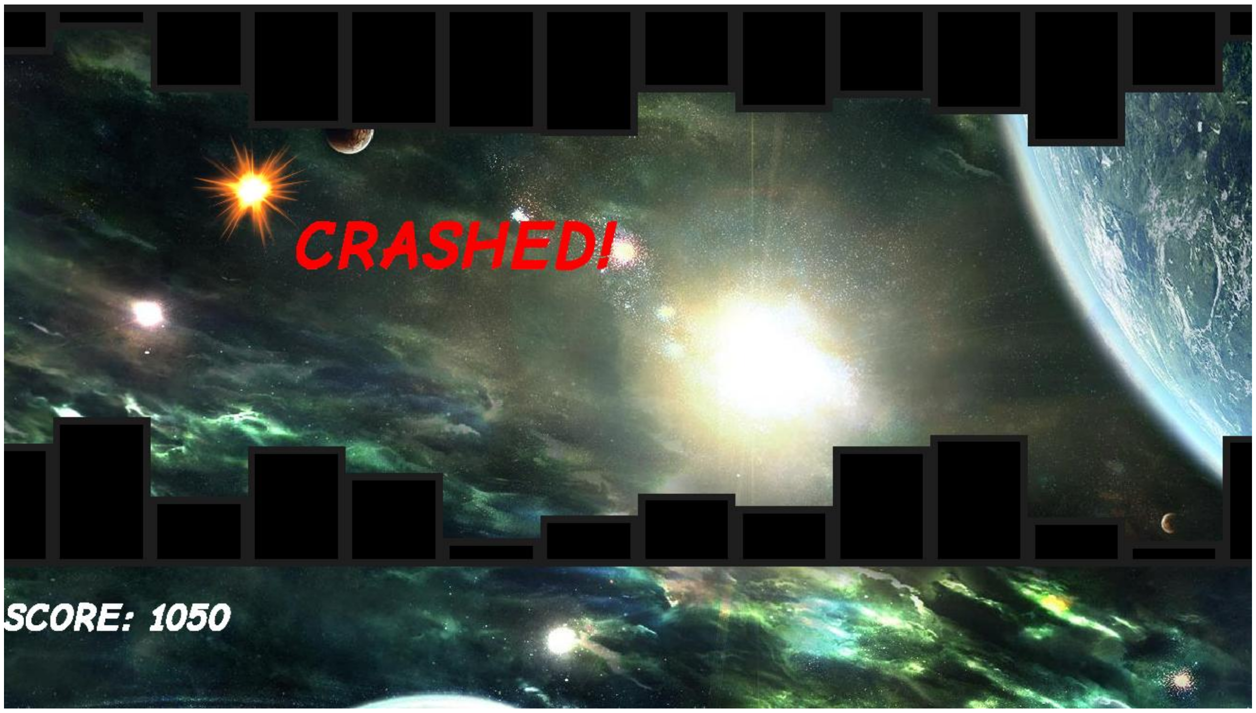
Demonstrating the Twitter Bird mode of the game. The blue circle represents a powerup



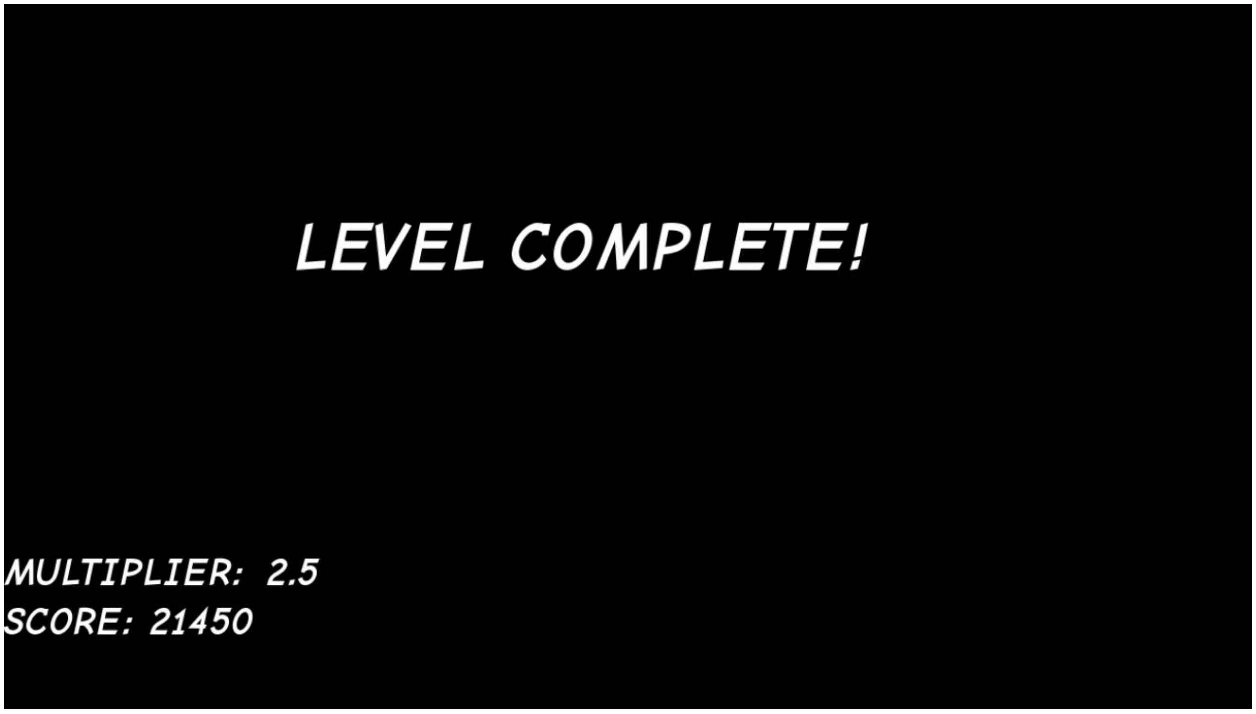
The ultimate level in the game!



The first level in the game



Demonstrating the crash indicator when the game detects a collision with the blocks



The screen showing when a level is complete

LIMITATIONS

The game take only runs in 720p resolution and hence may not run satisfactorily on higher resolution displays.

BIBLIOGRAPHY

The following websites and books were very useful towards the development of this project:

- stackoverflow.com
- cprogramming.com
- cplusplus.com
- sfml-dev.org
- hellointernet.fm
- Jumping into C++ by Alex Allain (book)

Development tools used:

- Geany IDE
- GNU g++/gcc toolset for Linux
- SFML 2.1 libraries