Bachelor's thesis

Information Technology

2014

Richard Olugbenga Ajayi

INTERACTIVE GAME PROGRAMMING WITH PYTHON (CODESKULPTOR)

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Networking

2014| 41 pages

Instructor: Patric Granholm

Richard Olugbenga Ajayi

INTERACTIVE GAME PROGRAMMING WITH PYTHON (CODESKULPTOR)

Over the years, several types of gaming platforms have been created to encourage a more organised and friendly atmosphere for game lovers in various works of life, culture, and environment.

This thesis focuses on the concept of interactive programming using Python. It encourages the use of Python to create simple interactive games applications based on basic human concept and ideas.

CodeSkulptor is a browser-based IDE programming environment and uses the Python programming language. Originally designed by Scott Rixner of Rice University, CodeSkulptor is based upon CodeMirror and Skulpt.

The method of carrying out the thesis was first to study the CodeSkulptor environment and the various modules available for the Python language and then delve into the development process. A total of five example games were written for this thesis.

KEYWORDS:

Gaming platforms, Python, browser-based programming, modules, CodeSkulptor, IDE.

CONTENTS

APPENDICES

FIGURES

TABLES

1 INTRODUCTION

1.1 Project Goal

The goal of this thesis is to demonstrate how CodeSkulptor is used for game programming. All of the program coding will be done via CodeSkulptor. CodeSkulptor is an online Python interpreter that runs on a browser. More details about CodeSkulptor will be discussed in chapter 3.

1.2 Project Scope

The scope of this thesis is to enable the development and implementation of the following topics:

- Programming using Functions, Logic, conditionals
- Event-driven programming, local and global variables, buttons and input fields
- Using canvas, static drawing, timers, interactive drawing
- List, keyboard input, motion positional/velocity control
- Acceleration and friction, spaceship class, sprite class, sound
- Sets, groups of sprite, collisions, sprite animation

1.3 Thesis Overview

Chapter 2: Computer programming languages

This chapter defines a program and discusses the types of programming languages. In addition, it introduces object-oriented languages and categories of programming languages.

Chapter 3: Programming Environments

This chapter sheds more light on the types of programming environments, IDEs and the CodeSkulptor environment. It also emphasizes more on the SimpleGui modules and how it can be applied in the reviewed games.

Chapter 4: Computer Games

This chapter highlights the process involved in the development of several games. The following games were reviewed and they include:

1. Rock-Paper-Scissors-Lizard-Spock:
   Randomly selects one of the choices and chooses a winner made on certain criteria's (see Chapter 4.1).
2. Guess The Number:
   A program randomly chooses a number and the player guesses the number chosen in a range of guesses.
3. Stopwatch:
   Time should be stopped exactly on a zero mark e.g. (should be x.xx.0)
4. Memory Game:
   A player picks cards/numbers with same number and finds all the 8 pairs to win in less number of turns.
5. RiceRocks (Asteroids):
   A ship hits as many asteroids as possible within 3 lives.

Chapter 5: Conclusion

This chapter concludes the entire thesis.

2 COMPUTER PROGRAMMING LANGUAGES

2.1 Programming languages

The collection of details instructions that is given when expecting a computer to perform a specific task is known as a program (Perry, G. 1994). Without detailed instructions, a computer is rendered useless. Computers cannot decide on their own to do what it wants except detailed instructions are given to it. This is where programming as a term is derived.

Programming languages are used to create programs that define computer behaviours and also provide the means for programmers to express computer algorithms. In other words, "a programming language is therefore a practical way for humans to give instructions to a computer" (Kioskea 2014). There are various types of programming Languages available for uses by both computers and people.

2.1.1 Machine language

A machine language is a low level programming language and it is complicated to read by humans but easily understood by computer. Processors use *machine code* which consists of 0s and 1s (Binary data). Codes written in this language are transformed into *machine code* for processing by the processor.

2.1.2 Assembly language

The "assembler" is very similar to the machine code but better understood by developers. Although assembly language statements are readable, they are still regarded as low-level programming languages.

One major disadvantage of an assembly language is that it is not portable from one machine to the other. The term "portability" describes the ability of a software program to be used on different types of machine.

2.1.3 High-level languages

In contrast to the machine and assembly languages, high-level languages "focus more on the programming logics rather than the underlying hardware components such as memory addressing and register utilization" (Janssen, C. 2013). These are languages that allow the development of a program in a simpler programming context regardless of the computer's hardware architecture. They are designed to make programming far easier and less error-prone.

Over the year various high-level languages have been developed which are closer to human language. Fortran II, one of the first languages, was introduced in 1958. In Fortran II the program would be written as; "C = A + B" which is quicker to write and more readable (O'Regan, G. 2012).

Many high level languages have existed since Fortran II, below are examples of widely used languages:

- Java
- C
- PHP
- Python
- C#
- JavaScript
- Perl

2.2 Object-oriented programming language

Object-oriented programming, or OOP, is a programming model organised around objects. Objects, which are usually specific realization of any object of classes, are used to communicate with one another to design applications and computer program. There are various examples of programming languages that contain object-oriented programming features; C++, C#, Java, Python, PHP, Ruby, etc.

Computer languages are regarded as object-oriented if the following fundamental concepts are included.

- Object:
  An object is an instance of a class.

- Class:
  A Class defines the characteristics of an object. These characteristics include attributes (fields or properties) and behaviours (methods or operations).

- Method:
  A method refers to the behaviour of an object and it affects one particular object within a program.

- Polymorphism:
  This is where one object can be treated and used like another object. Polymorphism allows two or more objects to respond to the same message.

- Inheritance is the ability to reuse existing objects where structure and methods in a particular class are passed down the hierarchy since classes are created in hierarchies. In other words, less programming is required when functions are added to complex systems.

- Encapsulation allows the separation of an object's implementation from its behaviour. It conceals the internal state of one object from other.

2.3 Interpretation and Compilation

There are two categories of programming languages which exist and some that make use of the combination of both categories. They are:

Interpreted languages
Complied languages
Intermediary languages

2.3.1 Interpreted language

This is a programming language that requires an external program called “the interpreter” which enables translation in order for the processor to understand the code. Codes written in this language are saved in the same format that they were entered (Vanguard 2012).

2.3.2 Compiled language

Programs written in complied language require and additional program called a “compiler”. This “compiler” creates a new stand-alone file (*set of machine specific instructions*) which requires no other program to execute itself. Compiled programs run faster that the interpreted ones because interpreted programs must be reduced to machine code at runtime (Vanguard 2012).

2.3.3 Intermediary language

Many languages make use of both the compiler and the interpreter (such as Pascal, C, BASIC, Python) as the programs written in these languages may undergo an intermediary compilation phase into a file where the source file is different from the language it was written in and still requiring an interpreter (non-executable) (Kioskea 2014).

3 PROGRAMMING ENVIRONMENTS

There are various programming environments available for programming but it is advisable for a learner to choose an environment that allows programs to be created and run at the same time. Programming environments can be broken down into two different types: integrated development environments and command-line environments (Eck, D. 2004).

Integrated development environments, or IDE, have been created over the years to help facilitate software development. An IDE, is a graphic user interface program that combines all aspects of programming and includes in most cases a debugger, a visual interface builder and project management. Some of these IDEs are built in such a way that they contain a compiler, interpreter or both as the case maybe (Eck, D. 2004). A command-line environment, on the other hand, is just a collection of commands that can be typed in to run programs, edit files, and compile source code.

3.1 Integrated Development Environments

There are various types of sophisticated IDE's available in various programming language and can be used for software development. Below are a few:

- Eric -- an IDE written in Python with PyQt and contains some familiar Qt tools such as Designer and Linguist (Eric 2014).
- Eclipse IDE -- a very popular professional development tool that supports Java development (The Eclipse Foundation 2014).
- IDLE -- a Python IDE built with the Tkinter GUI toolkit and comes with most distributions of Python (Python Software Foundation 2014).
- NetBeans IDE – a free "open source" program that should run on any system with Java 1.3 or later (Oracle Corporation 2013)
- JCreator -- for Windows and a lighter-weight IDE that works on top of Sun's SDK (Xinox Software 2014).

- CodeSkulptor – a browser based IDE developed for creation of Python GUI apps using a self-maintained library: SimpleGui. Applications written can be saved and shared via a URL to friend or colleagues.

3.2 CodeSkulptor

CodeSkulptor is a browser-based programming environment for programming in the Python language and it implements a subset of Python2. In addition, three graphical libraries were added. These libraries include: SimpleGui, SimpleMap and SimplePlot.

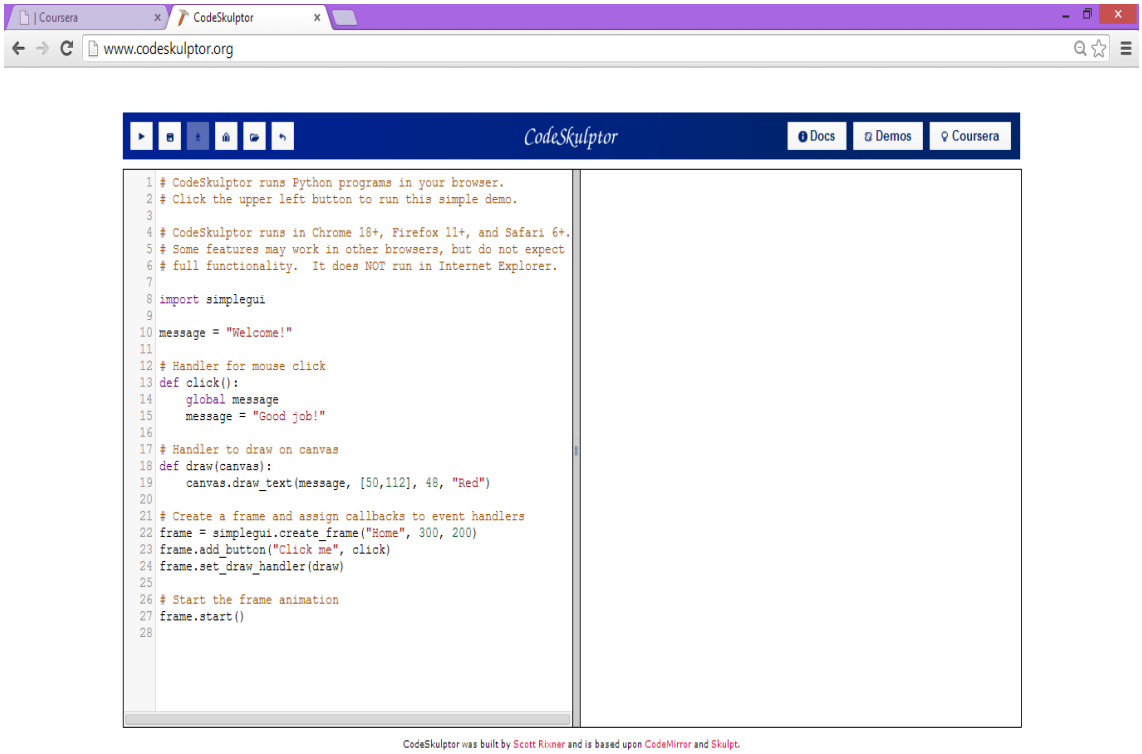

Figure 1. CodeSkulptor Environment

The figure below shows the CodeSkulptor environment (*can be opened via www.codeskulptor.org*) which is made of several sections. They are:

- Control area (upper Left) - This area contains buttons that allows the control of the application

Figure 2. Control Area

- Editor (Lower Left) - Python programs are written directly in this area.
- Console area (Lower right) – This is where the textual output of the program are displayed
- Documentation area (Upper right) – This area allows the user access to all documentations, videos and demonstrations that facilitate the use of CodeSkulptor. The “Coursera” button links the user/the developer to the course page on coursera’s site. It contains examples and exercises.

Figure 3. Documentation Area

Chrome 18+, Mozilla Firefox 11+, Apple Safari 6+ are the recommended browsers used for CodeSkulptor. Microsoft Internet Explorer has poor functionality (Warren et al. 2014).

3.3 Modules

If the Python interpreter is exited and then re-launched again, all the functions and variable definitions are lost. In order to avoid such occurrence in cases where longer programs are written, a script is created using a text editor to prepare input for the interpreter and running it with that file as input.

Such occurrence as described in the above paragraph is supported by Python, whereby definitions (functions and variables) are put in a file and used in an interactive instance of the interpreter. Such a file is referred to as a module.

”A module is a file containing Python definitions and statements”
(Python Modules 2014).

3.3.1 Standard module

The following subset of Python standard library was implemented in CodeSkulptor. These modules are imported using the import statement

Math Module:

This module contains mathematical operations and they contain several operations like;

- Trigonometric Functions (*math.sin(), math.cos(), math.tan()*),
- Exponentiation (*math.pow(), math.exp()),* Square Root (*math.sqrt()*),
- Logarithm (*math.log()*), etc.,

Syntax:

math.pow (x, y),
math.pi

Table 1. Math module examples

Code	Output
import math print math.pow(2, 3)	8
import math print math.pi	3.141592653589793

Random Module:

This module contains functions that involve randomness and they include:

- Random integer (*random.randint(), random.randrange()*),
- Permute List (*random.shuffle()*),
- Random Element of Sequence (*random.choice()*), etc.,

Syntax:
random.choice(a_seq)
random.shuffle(a_list)

random.randrange(start, stop, step)

Table 2. Random choice, randrange examples

Code	Output
import random print random.choice([1, 2, 3, 4, 5])	#result is one of the sequence's element
import random print random.randrange(0, 10, 2)	#results are: 0, 2, 4, 6, 8

3.3.2 Graphical module

There are three (3) custom modules implemented by CodeSkulptor for graphics in a browser. The appropriate module is imported using the import statement as described in the previous module. The following graphical modules are described below:

SimpleGUI module:

This is specifically used for interactive games and drawings. It was written in JavaScript allowing users to use the web browser. Most of the games featured in this thesis use the SimpleGui module (Warren, J. et al. 2014).

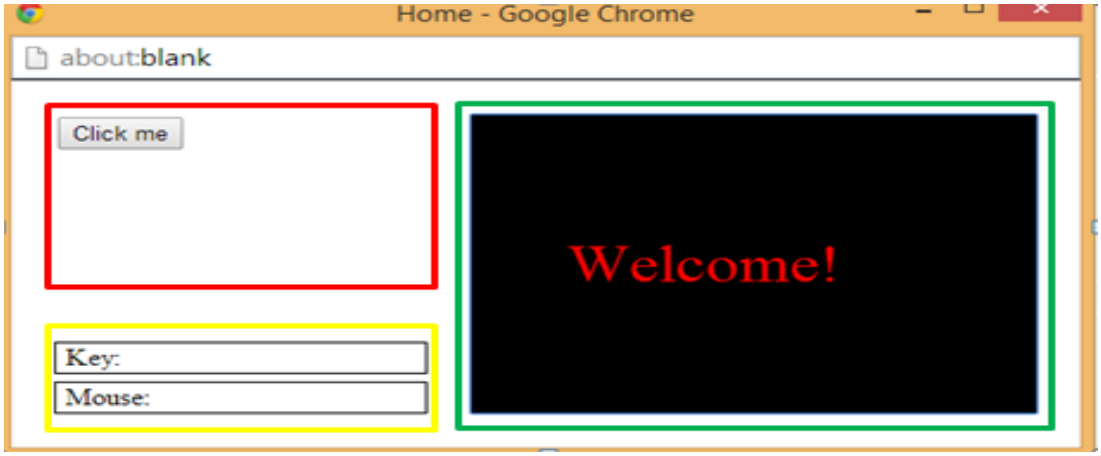

Figure 4. SimpleGui frame

The SimpleGui frame is broken into 3 parts as seen in Figure 4 and they include:

1. Control Area (Red colour)
   It contains buttons, text inputs and text labels that allow printing of status information from the application back to the frame.
2. Status Area (Yellow colour)
   It gives feedback about keyboard and mouse events that happen in the canvas and aids debugging simpler in SimpleGui.
3. Canvas (Green colour)
   The interesting part of the frame where images, text, shapes are drawn and respond to keyboard and mouse events

SimpleMap Module:

This particular module is used for drawing map features and contains several superclass objects. They include: Maps, Markers and Lines.

SimplePlot module:

This is used for plotting of numeric data.

3.3.3 Other module

Numeric Module:

This module is not a part the standard Python but a smaller module just like the common NumPy module (Numpy developers 2014). It is one of the modules used for mathematical with two-dimensional matrices and provides basic linear algebra operations.

CodeSkulptor module:

This is a small collection of miscellaneous operations not available in standard Python.

4 COMPUTER GAMES

4.1 Rock-Paper-Scissors-Lizard-Spock(RPSLS)

This is an amplified version of the hand game “Rock-paper-scissors” but this particular game allows five choices. The rule is that any choice loses to its clockwise opponent and beats its counter-clockwise opponent.

In this case, each name is represented by a number form 0 – 4. Example:

0 - rock
1 - Spock
2 - paper
3 - lizard
4 - scissors

Scissors cut paper.
Paper covers rock.
Rock crushes lizard.
Lizard poisons Spock.
Spock smashes scissors.
Scissors decapitate lizard.
Lizard eats paper.
Paper disproves Spock.
Spock vaporises rock.
Rock crushes scissors.

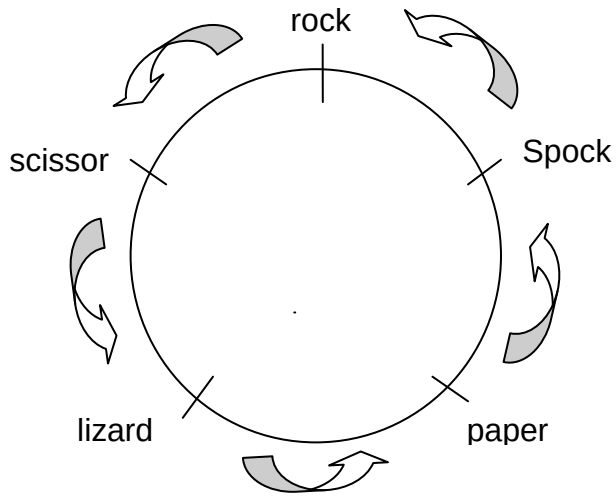

Figure 5. RPSLS counter-clockwise diagram

Development process:

- A helper function that converts numbers between 0 - 4 into string (names):

```
def number_to_name(number):
    if number == 0:
        name = "rock"
```

```
    elif number == 1:
        name = "Spock"
    elif number == 2:
        name = "paper"
    elif number == 3:
        name = "lizard"
    elif number == 4:
        name = "scissors"
    else:
        print "#error: No name associated with the number"
    return name
```

- A helper function that converts strings(names) into number between 0 and 4 is coded:

```
def name_to_number(name):
    if name == "rock":
        number = 0
    elif name == "Spock":
        number = 1
    elif name == "paper":
        number = 2
    elif name == "lizard":
        number = 3
    elif name == "scissors":
        number = 4
    else:
        print "#error: No number associated with name"
    return number
```

- The first part of the main function rpsls(name) is constructed. It uses the helper function name_to_number to convert name into player number.

```
def rpsls(name):
        player number = name_to_number(name)
            ….
```

- The second part of the main function rpsls(name) randomly generates a number (comp number) between 0 – 4 using one of the random module described in chapter 4 (random. Range()).

```
def rpsls(name):
        …
        comp_number = random.randrange(0,5)
        …
```

- The final part of the main function rpsls(name) determines the winner and displays it via the print method.

```
def rpsls(name):
                …
                …
        difference = player_number - comp_number
        difference_modulo_five = difference % 5
```

The difference is calculated as seen and a remainder operation (%) is used to determine the distance between the comp_number and player_number.

The winner is calculated thus:

```
                …
if 1 <= difference_modulo_five <= 2:
        print "Player wins!"
elif 3 <= difference_modulo_five:
        print "Computer wins!"
elif difference_modulo_five == 0:
```

```
                print "Player and computer tie!"
        …
```

Output:

```
rpsls("rock")
rpsls("Spock")
rpsls("paper")
rpsls("lizard")
rpsls("scissors")
```

```
Player chooses rock
Computer chooses rock
Player and computer tie!


Player chooses Spock
Computer chooses rock
Player wins!


Player chooses paper
Computer chooses scissors
Computer wins!


Player chooses lizard
Computer chooses Spock
Player wins!


Player chooses scissors
Computer chooses rock
Computer wins!
```

Figure 6. Console output for RSPLS

4.2 Guess the number

This game is a simple two-player game. The concept, however, is for one player to think of a secret number and the second player to guess it. A range is drawn initially so the guesser can have an idea of the range to guess from.

To make the game much more appealing, the thinker tells the guesser if he should go “higher, lower or correct!” if the guess is right. The guesser has to

make an attempt to guess right before he uses all available guesses. E.g. “7” or “10” guesses for a range between “0 – 100” and “0 – 1000” respectively.

For this game, “Guess your number”, we will be using one of the graphics modules: simpleGui and two standard modules: math and random module as described in chapter 3

Development process:

- Initialize and decide global variables that control the state of the game. E.g. Number generated by the program, the guess number that assigns the number of guesses for each game range.

```
low = 0
high = 100

global number
global guessNumber
```

- Generate random secret number in a giving range. This is generated between a giving range, low or high.

```
number = random.randint (low, high)
```

- In other to create an input text box, the simpleGui module is imported.

```
import simplegui
```

- Take the input “guess” from the user by writing the event handler. The function “input_guess(guess)” compares the guess to the secret number and prints out an appropriate message as written in the code.

```
def input_guess(guess):
```

```
global guessNum

guessNum -= 1
guess = int(guess)

print ""
print "Guess was", guess
print "Number of remaining guesses is", guessNum

if guessNum > 0:
        if guessNum >= 1:
                if guess == number:
                print "Correct!"
                print""
                new_game()
            elif guess > number:
                print "Lower!"
            elif guess < number:
                print "Higher!"
elif guessNum == 0 and guess == number:
        print "Correct!"
        print""
        new_game()
else:
      print "You ran out of guesses. The number was",
 number
      print ""
      new_game()
```

- Create a function “new_game()”. This function computes the generation of the secret number as well as the number of guesses for each range and assigns it to global variables.

```
def new_game():
        ***find code in appendix 1***
```

Buttons and input field

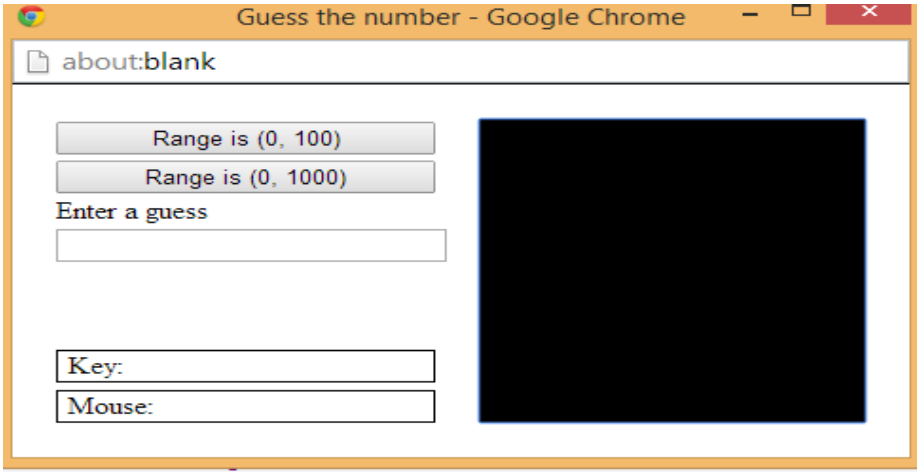

Figure 7. Guess the number frame

To make the game more entertaining, two buttons were created. These buttons allow the player to switch between two different ranges: “0 – 100” and “0 – 1000”. Each button resets the secret number and prints an appropriate message informing the player of the range in which the guess should be made.

```
# register event handler for control elements
frame.add_button("Range is (0, 100)", range100, 200)
frame.add_button("Range is (0, 1000)", range1000, 200)
```

The input field registers the guess made by the player. Each guess is automatically registered after the “Enter Key” is pressed.

```
# register event handler for control elements
frame.add_input("Enter a guess", input_guess, 200)
```

Output:

The final outcome of the output is printed on the console:

```
New game. Range is from 0 to 100
Number of remaining guesses is 7

Guess was 55
Number of remaining guesses is 6
Lower!

Guess was 35
Number of remaining guesses is 5
Lower!

Guess was 25
Number of remaining guesses is 4
Higher!

Guess was 20
Number of remaining guesses is 3
Higher!

Guess was 30
Number of remaining guesses is 2
Lower!

Guess was 32
Number of remaining guesses is 1
Lower!

Guess was 29
Number of remaining guesses is 0
Correct!
```

Figure 8. Guess the number console output

4.3 Stop Watch game

This project involves the building of a simple digital stopwatch that keeps track of time in tenths of a second. For this game, we will focus more on combining text drawing in the canvas.

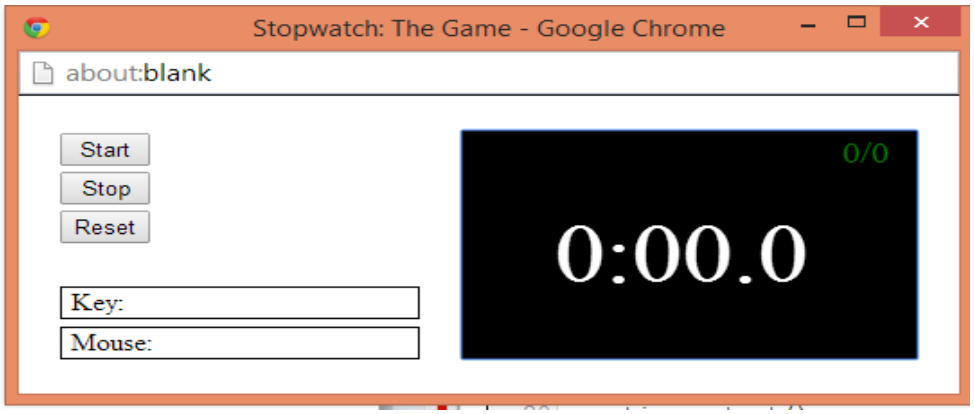

Figure 9. Stopwatch game frame

The stopwatch in Figure 9 contains three buttons: Start, Stop and Reset.

In addition, two green zeros with a slash in between located at the top right of the canvas. This area of the game was used to test the reflexes of the player and used to show the number is “hits” per “tries”. The number of “hits” can be

explained further as the number times the player hits a zero mark, i.e., 0:02.0, 0:15.0, etc. while the number of “tries” is the number of times the game was played.

Development process:

- Firstly a timer is created whose event handler is incremented by a global integer. The timer has an interval of 0.1 sec.

```
# define event handler for timer with 0.1 sec interval
def timer():
        global t
        t+=1

# register event handlers
timer = simplegui.create_timer(100, timer)
```

- Draw the current time in the middle of the canvas by using an event handler function

```
# define draw handler
def draw(canvas):
    canvas.draw_text(format(t), (50, 100), 56, "White")
    canvas.draw_text(str(wins) + "/" + str(tries), (210,20), 20,
"Green")
```

- Add “Start: to start the time”, “Stop: to stop the time” and “Reset: stops and resets the current time to zero” buttons.

```
def start():
    ***find code in appendix 2***

def stop():
```

```
    ***find code in appendix 2***

def reset():
    ***find code in appendix 2***

# register event handlers
start = frame.add_button("Start", start, 50)
stop = frame.add_button("Stop", stop, 50)
reset = frame.add_button("Reset", reset, 50)
```

- Write a helper function **format(t)** that returns a string in the form of “A:BC:D”. String A, C and D are digits in the range 0-9 and B is between 0-5.

```
def format(t):
    a = t // 600
    b = ((t // 100) % 6)
    c = ((t // 10) % 10)
    d = t % 10
return str(a) + ":" + str(b) + str(c) + "." + str(d
```

4.4 Memory Game

This is a simple card game where a player is expected to pick cards with the same number in pairs. In one turn, two of the cards are flipped over. If they match, they are left revealed by keeping them face up. If they do not match, both cards are flipped back down. The aim behind the game is to match all the pairs of cards flipped facing up in the minimum number of turns.

Development process:

- Create a deck of cards as a list consisting of 16 numbers. Each number should lie in the range(0,8) and appearing twice.

```
card_List = [i % 8 for i in range(16)]
```

- Create a draw handler that iterates through the **card_List** using a for loop and draws the numbers on the canvas

```
def draw(canvas):
    ….
    a = -35
    for i in range(len(card_List)):
        a += 50
        canvas.draw_text(str(card_List[i]), [a, 60], 35, "White")
```

- Shuffle the cards using *random.shuffle()* module

```
random.shuffle(card_List)
```

- Modify the draw handler created in second list, of the development process, to draw a bank Green rectangle (or colour of your choice) or card value.

```
def draw(canvas):
    ……..
    ……..
    ……..
    b = -50
    for i in exposed:
        b += 50
        if not i:
            canvas.draw_polygon([(b, 0), (b + 50, 0), (b + 50, 100), (b +
0, 100)], 2, "White", "Green")
```

For proper implementation of this behaviour, a second list should be created called exposed.

```
exposed = [False for i in range(16)]
```

In the exposed list,

the i^{th} entry = True if the card faces up and its value visible

the i^{th} entry = False if the card faces down and its value hidden

by default all entries are set to False.

- Create an event handler **mouseclick** and modify it to flip cards based on the location of the mouse click, whereby, a players click on the i^{th} card and the value of exposed[i] be changed from “False” to “True”.

```
def mouseclick(pos):
```

There are 3 states to the mouse click:

First state (counter = 0): The card is clicked to expose the card value (*False to True*) and the counter is increased 1.

Second state (counter = 1): The 2^{nd} card is clicked to expose the 2^{nd} card value (*False to True*) and the counter is increased 1. If exposed card are clicked it mouseclick is ignored

Third state (counter = 2): The 3^{rd} card list is clicked to expose the card value (*False to True*) and the counter is decreased 1. If the 1^{st} and 2^{nd} cards are not equal they return back False (Hidden) and your back again at the first state.

It goes on to the last until all cards are appended.

```
if counter == 0:
    card.append(pos[0]/50)
    exposed[pos[0]/50] = True
    counter += 1
    turns = 1

elif counter == 1:
```

```
            if not (pos[0]/50 in card):
                card.append(pos[0]/50)
                counter += 1
            exposed[pos[0]/50] = True

        elif counter == 2:
            if not (pos[0]/50 in card):
                if card_List[card[-1]] != card_List[card[-2]]:
                    exposed[card[-1]] = False
                    exposed[card[-2]] = False
                    card.pop()
                    card.pop()
                counter -= 1
                turns += 1
                exposed[pos[0]//50] = True
                card.append(pos[0]//50)
```

- A counter is added to keep track of the number of turns. During the first state the number of turns is incremented by 1 and also at the third state.

```
def draw(canvas):
    label.set_text("Turns = " + str(turns))
```

- Implement a **new_game()** function that restarts the game via the “Reset button”

```
def new_game():
    global card_List, exposed, card, counter, turns
    ***see appendix 3***

frame.add_button("Restart", new_game)
```

Output:

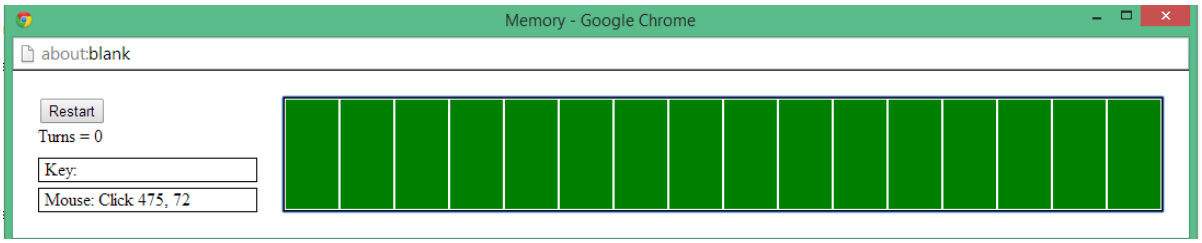

Figure 10. Guess the number - Initial screen

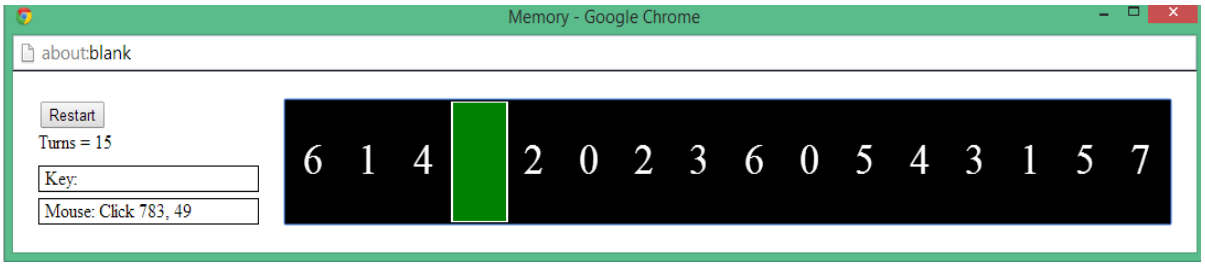

Figure 11. Guess the number - playing mode

4.5 RiceRocks (Asteroids) game

The game RiceRocks is an implementation of the classic game, Asteroids, released in November 1979. The object of the game is to shoot and destroy asteroids (rocks) while the player avoids colliding or being hit by them.

This game uses event-driven programming to handle ”clicks” and ”pressing / releasing” from the mouse and keyboard respectively. Objects for the game are drawn on the canvas by a draw procedure. This procedure runs at a rate of 60 times per second, updating object position and redraws them in a new position (Al-hindi, M. 2013).

In other to make the development of this game possible a ricerocks program template (*check references*) was made available. The following were implemented to bring out the game:

i. To make sure multiple rocks are spawn randomly and at different points of the game and all rocks spawn on the canvas must not be greater than 12 rocks. If a particular rock is destroyed or at a point it moved out of the

screen, a new one will be spawn. A rock must not spawn at the position on the ship.

ii. Explosion should occur when:
    a. Missile hits rock (a rock set is deletes from the set)
    b. Rock hits ship or vice versa.

iii. Scores accumulated and the number of lives should be updated each time and explosion occurs. When score is equal to zero, the splash screen appears and the game is reset to its initial state.

iv. Each missile should have a lifespan so it does not keep looping forever in the game.

Figure 12. RiceRocks (Asteroids) - Splash screen

Development process:

Multiple Rocks

- Initialize the rock group to an empty set and program the **rock_spawner** function to create a new rock and add it to **rock_group**

```
rock_group = set([])

# timer handler that spawns a rock
```

```
def rock_spawner():
    global rock_group
    rock_pos = [random.randrange(0, WIDTH), random.randrange(0,
            HEIGHT)]
    rock_vel = [random.random() * .6 - .3, random.random() * .6 - .3]
    rock_avel = random.random() * .2 - .1
    a_rock = Sprite([WIDTH * random.random(), HEIGHT *
            random.random()], [random.random() * 3 -
            1.5,random.random() * 3 - 1.5], 0, (random.random() -
            .5) / 8, asteroid_image, asteroid_info)
    if len(rock_group) < 12 and started == True:
        if dist(a_rock.pos, my_ship.get_position()) > 150:
            rock_group.add(a_rock)
```

- Create a helper function **process_sprite_group** which is responsible for calling update and draw methods for each sprite in the group

```
#updates sprites and removes them at the end of lifespan
def process_sprite_group(a_set, canvas):
    for sprite in a_set:
        if sprite.update() == False:
            a_set.remove(sprite)
        sprite.draw(canvas)
```

- Call the **process_sprite_group** function in the draw handler on **rock_group**

```
# draw ship and sprites
process_sprite_group(rock_group, canvas)
```

Collisions

- A **collide** method is added to the Sprite class which takes and argument “other_object”. If there is a collision “True” is returned otherwise “False”

```
# Sprite class
```

```
class Sprite:
    .......
    def collide(self, other_object):
        if dist(self.get_position(), other_object.get_position()) <=
                (self.radius + other_object.radius):
            return True
        else:
            return False
```

- The helper function **group_collide** is implemented. This function takes 2 arguments: (a set **group** and a sprite **other_object**) and checks for collision between the elements of the **group** and **other_group**. The collided object is removed from the group is there is a collision detected.

```
#checks for collisions between one set and one object
def group_collide(group_set, other_object):
    remove_collided = set([])
    for item in group_set:
        if item.collide(other_object):
            remove_collided.add(item)
            new_explosion = Sprite(item.pos, (0,0), 0, 0,
                explosion_image, explosion_info, explosion_sound)
            explosion_group.add(new_explosion)
    if len(remove_collided) > 0:
        group_set.difference_update(remove_collided)
    return len(remove_collided)
```

- Include in the draw handler the helper function **group_collide**. If a ship hits any of the rock, the number of lives is reduced by one.

```
if group_collide(rock_group, my_ship) > 0:
        explosion_group.add(Sprite(my_ship.get_position(), [0, 0], 0,
                0, explosion_image, explosion_info))
```

```
lives -= 1
```

Missiles

- Initialize the missile group to an empty set and modify the shoot method of the ship class to create new missile and add it to the **missile_group.** The firing sound should play each time a missile is spawned.

```
missile_group = set([])
```

- Use the **process_sprite_group** helper function to process the **missile_group** in the draw handler.

```
process_sprite_group(missile_group, canvas)
```

- Every time update is called there is an increment in the age of the sprite, in the **update** method of the Sprite class. False is returned if the age is less than the lifespan and True otherwise.

```
def update(self):
    # update angle
    self.angle += self.angle_vel

    # update position
    self.pos[0] = (self.pos[0] + self.vel[0]) % WIDTH
    self.pos[1] = (self.pos[1] + self.vel[1]) % HEIGHT

    #update age
    self.age += 1
    if self.age < self.lifespan:
       return True
    else:
       return False
```

The aim for the missiles is to destroy rock. Due to this fact, a helper function **group_group_collide** is implemented that takes two groups as input. It returns the number of elements in the first group that collides with the second and deletes these elements in the first.

Game Control

The spaceship is controlled using various keys on the keyboard as defined by the game programmer. Two key handlers: **keydown(key)** and **keyup(key)** to help in controlling the spaceship were created. For this particular game, the spacebar is used to shoot missiles, ”left/right” arrow keys on the keyboard rotates spaceship, while the ”up” arrow key is used for acceleration.

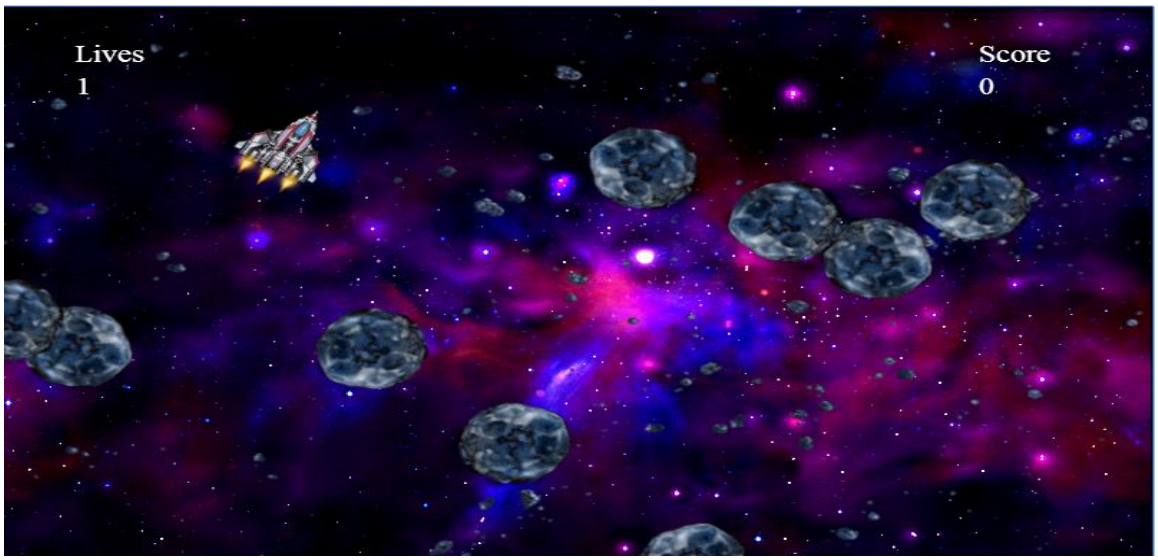

Figure 13. RiceRocks (Asteroids) - Play Mode

5 CONCLUSION

In the introduction of this thesis, the project goal and scope were highlighted and specific detail of the implementation of several listed topics were listed. CodeSkulptor has served and accomplished the purpose of game implementation. Most of the game development processes mentioned are reusable over and over again in so many ways to accomplish a better gaming programming experience.

In this thesis, the author successfully implemented 5 games (See Chapter 4 of this thesis). The games can be accessed and played via links provided in the References *(pg. 38 – 40)*. All the games were developed using the CodeSkulptor environment. Out of the three graphical modules mentioned in Chapter 3, SimpleGUI was the only one used in all the games implemented because it is specifically used for interactive games and drawing.

Interactive game programming using Python (CodeSkulptor) has helped to demonstrate how several topics like functions, conditional statement, local/global variables, buttons/input field, using canvas, motion controls, sprite class, sprite animation, sound, groups of sprite, sets, etc., can be written to give a great game programming experience.

REFERENCES

Al-hindi, M. 2013, , *RiceRock Asteroids*, “Retrieved: 20 May 2014” http://python-alhindi.blogspot.fi/2013/12/mini-project-8-ricerocks-asteroids.html edn.

Asteroids 2014, , *Asteroids*. “Retrieved: 20 May 2014” http://en.wikipedia.org/wiki/Asteroids_(video_game).

Eck, D. 2004, , *Introduction to Programming Using Java*. “Retrieved: 12 October 2014” http://math.hws.edu/eck/cs124/javanotes4/progenv.html.

Eric 2014, , *Eric Python IDE*. “Retrieved: 12 October 2014” http://eric-ide.python-projects.org/index.html [2014, .

Fruit, J. 2013, , *Comparison of Python IDEs for Development*. “Retrieved: 11 September 2014” http://pythoncentral.org/comparison-of-python-ides-development/.

Game1 2013, , *RPSLS*. “Retrieved: 11 September 2014” http://www.codeskulptor.org/#user20_3P2RlV5wIuMQg54.py.

Game2 2013, , *Guess the number*. “Retrieved: 11 September 2014” http://www.codeskulptor.org/#user21_wOQOKjw6zCRFkiX.py.

Game3 2013, , *Stop Watch*. “Retrieved: 11 September 2014” http://www.codeskulptor.org/#user22_VJnCieTAvXDodwL.py.

Game4 2013, , *Memory*. Retrieved: 11 September 2014 http://www.codeskulptor.org/#user24_QhHeGE4Ogc_0.py.

Game5 2013, , *Ricerocks Game*. Retrieved: 11 September 2014 http://www.codeskulptor.org/#user27_QDvK0PYBnbuT0zu.py.

Giampaolo, R. 2014, , *GUI*. “Retrieved: 20 May 2014” https://wiki.python.org/moin/UsefulModules#GUI.

Greiner, J. 2012, , *CodeSkulptor documentation*. “Retrieved: 20 May 2014” http://www.codeskulptor.org/docs.html#tabs-Python.

IDE 2014, , *Integrated development environment*. “Retrieved: 12 October 2014” http://en.wikipedia.org/wiki/Integrated_development_environment.

Interpreted Language 2014, , *Interpreted language*. “Retrieved: 20 May 2014” http://en.wikipedia.org/wiki/Interpreted_language.

Janssen, C. 2013, , *High-Level Language (HLL)*. “Retrieved: 20 May 2014” http://www.techopedia.com/definition/3925/high-level-language-hll.

Kioskea 2014, , *Programming Languages*. “Retrieved: 20 May 2014” Available: http://en.kioskea.net/contents/315-programming-languages.

Lucas 2012, , *INFORMATION TECHNOLOGY, INFORMATION SYSTEMS, ETC.* “Retrieved: 20 May 2014” http://it-is-etc.blogspot.fi/2012/12/python-interactive-programming.html.

Numpy developers 2013, , *Numpy module*. “Retrieved: 20 May 2014” http://www.numpy.org/.

O’Regan, G. 2012, *A Brief Histroy of Computing,* Second edn, .

Oak, M. 2008, , *Programming Languages*. “Retrieved: 20 May 2014” http://www.buzzle.com/articles/list-of-programming-languages.html.

Oracle Corporation 2013, , *NetBeans IDE*. “Retrieved: 11 September 2014” https://netbeans.org/.

Perry, G. 1994, *Absolute Beginner’s Guide to C ,* Second edn, Richard K. Swadley.

Python Documentation 2014, , *Python 3.4.1 documentation*. “Retrieved: 20 May 2014” https://docs.python.org/3/.

Python Game Libraries 2013, , *Python Game Libraries*. “Retrieved: 20 May 2014” https://wiki.python.org/moin/PythonGameLibraries.

Python Modules 2014, , *Modules*. “Retrieved: 20 May 2014” http://docs.python.org/2/tutorial/modules.html#.

Python Software Foundation 2014, , *IDLE*. “Retrieved: 10 October 2014” https://docs.python.org/2/library/idle.html.

Qt 2014, , *Qt-Project*. “Retrieved: 10 October 2014” http://qt-project.org/ [2014, .

Rao, A. 2012, , *My projects in Codeskulptor*. “Retrieved: 20 May 2014” http://www.ak-droid.com/my-projects-in-codeskulptor/.

Ricerocks 2012, , *Ricerocks program template*. “Retrieved: 11 September 2014” http://www.codeskulptor.org/#examples-ricerocks_template.py.

Riverbank Computing Limited 2014, , *PyQt*. “Retrieved: 10 October 2014” http://www.riverbankcomputing.co.uk/software/pyqt/intro [2014, .

Rizos, Y. & Badar 2012, , *stack exchange, inc;*. “Retrieved: 20 May 2014” http://programmers.stackexchange.com/questions/17976/how-many-types-of-programming-languages-are-there.

The Eclipse Foundation 2014, , *Eclipse*. “Retrieved: 10 October 2014” https://www.eclipse.org/ [2014, .

Tkinter 2014*, Tkinter GUI toolkit*. “Retrieved: 10 October 2014”Available: http://en.wikipedia.org/wiki/Tkinter.

Vanguard Software Corporation 2012, , *Compiled vs. Interpreted Languages*. “Retrieved: 20 May 2014” http://www.vanguardsw.com/dphelp4/dph00296.htm.

Warren, J., Rixner, S., Greiner, J. & Wang, S. 2014, , *An Introduction to Interactive Programming in Python*. “Retrieved: 20 May 2014” http://www.codeskulptor.org/#examples-simplegui-0.py.

Xinox Software 2014, , *JCreator IDE*. “Retrieved: 10 October 2014” http://www.jcreator.com/.

APPENDICES

Appendix 1. Helper function to start and restart the game (Guess the number)

```
def new_game():
    global number
    global guessNum

    number = random.randint(low,high)
    guessNum = int(math.ceil(math.log(high - low + 1, 2)))
    print ""
    print "New game. Range is from", low, "to", high
    print "Number of remaining guesses is", guessNum
    print ""
```

Appendix 2. Event handlers for buttons; "Start", "Stop", "Reset" (Stop watch)

```
def start():
    timer.start()


def stop():
    global wins
    global tries
    if not timer.is_running():
        return
    timer.stop()
    tries += 1
    if t % 10 == 0:
        wins += 1

def reset():
    global t
    global wins
    global tries
    t = 0
    wins = 0
    tries = 0
    timer.stop()
```

Appendix 3. Helper function to initialize globals (Memory game)

```
def new_game():
    global card_List, exposed, card, counter, turns
    card_List = [i % 8 for i in range(16)]
    random.shuffle(card_List)
    exposed = [False for i in range(16)]
    card = []
    counter = 0
    turns = 0
```