

A2 F454 Computing Project

adventureCircle

Contents

• <u>Definition, Investigation and Analysis</u>	
➤ Definition – nature of the problem to be investigated-----	3
➤ Investigation and Analysis-----	5
• <u>Design</u>	
➤ Nature of the solution-----	22
➤ Algorithms-----	41
➤ Test strategy-----	48
• <u>Software Development and Testing</u>	
➤ Software development-----	54
➤ Testing-----	76
• <u>Documentation</u>	
➤ Documentation-----	87
• <u>Evaluation</u>	
➤ Discussion... in meeting the original objectives-----	98
➤ Evaluate the user’s response to the system-----	101
➤ Desirable extensions-----	101

(a) *Definition, Investigation and Analysis*

(i) **Definition – nature of the problem to be investigated**

Background

I am going to develop a game that fuses a little bit of the classic games “Pong” and “Snake”. Pong is a game where the player controls an in-game paddle by moving it vertically across the left side of the screen, and can compete against either a computer-controlled opponent or another player controlling a second paddle on the opposing side. Players use the paddles to hit a ball back and forth. The aim is for each player to reach eleven points before the opponent; points are earned when one fails to return the ball to the other. Snake is a game where the player moves a snake-like sprite across a plane which increases in length as it consumes items. The more it consumes the longer it gets and the bigger an obstacle to itself it becomes. The game I want to create is like Pong because of the simplicity of the design, for example a black and white colour scheme, and the simplicity of the game itself. It is like Snake because of the interactions with items and how that affects the original Sprite. My game will be an object interaction game like my two examples.

The screens below illustrate the current games that I’m going to investigate:

Figure 1 – An image of the original game of Pong, first developed and played in 1972

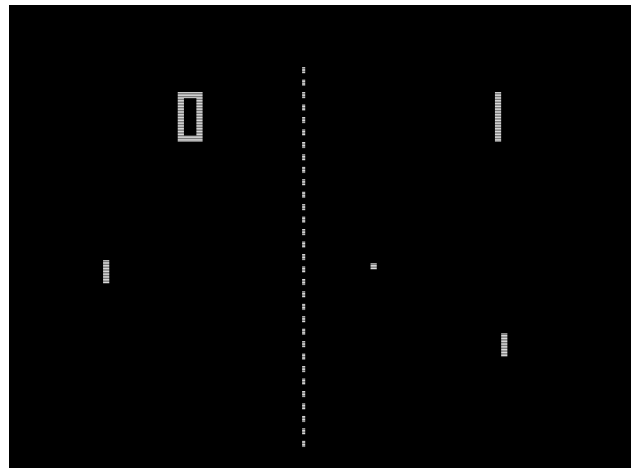
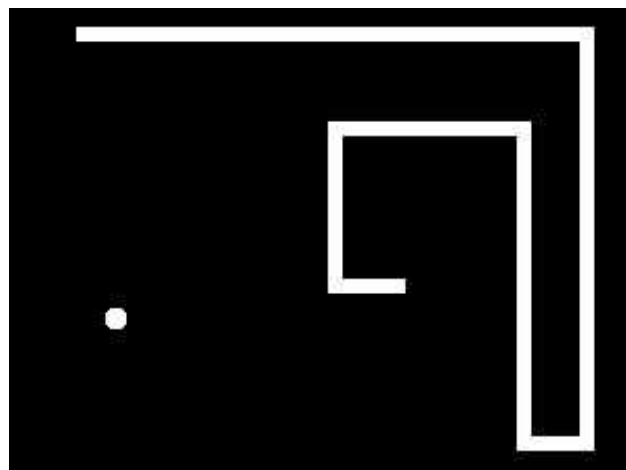


Figure 2 – An image of a simplified version of Snake. The original game was first developed in 1976 with a different concept but the actual “Snake” game was popularised by Nokia phones from 1998



Problem

At this current point, I believe that the two games, though classics, are outdated and not played as much as they were before. The enjoyable factor of the games has subsided substantially. I believe that I can make a new game that is more current and in trend and be enjoyable to play. I'd bring the best features of the two games together to create a fun game with a new concept. I observed this problem through my brother (who is 9 years old) at home, who is a prime example of my end user target. I observed that he does not play any games like Snake or Pong as he feels he has tried all versions of these types of games. I think the new concept of my game will appeal to him and other children around his age. To make my game appeal to that kind of target audience I am considering redesigning the type of interface to make it more child user friendly. The full scope of the problem is not yet known and will require further investigation to discover.

End User

My end user target will be primary school children (7-11 year olds), who I believe will enjoy my game the most as they will not have as much experience with these type of games as people my age. The group that I will be referring to throughout the process for opinions/improvements/testing will be my brother (9 years old) and four of his friends (8-9 years old).

Computational Method (justification):

The problem can be resolved using computational methods such as algorithms, coding, flowcharts and engineering principles among other things to bring the system to fruition.

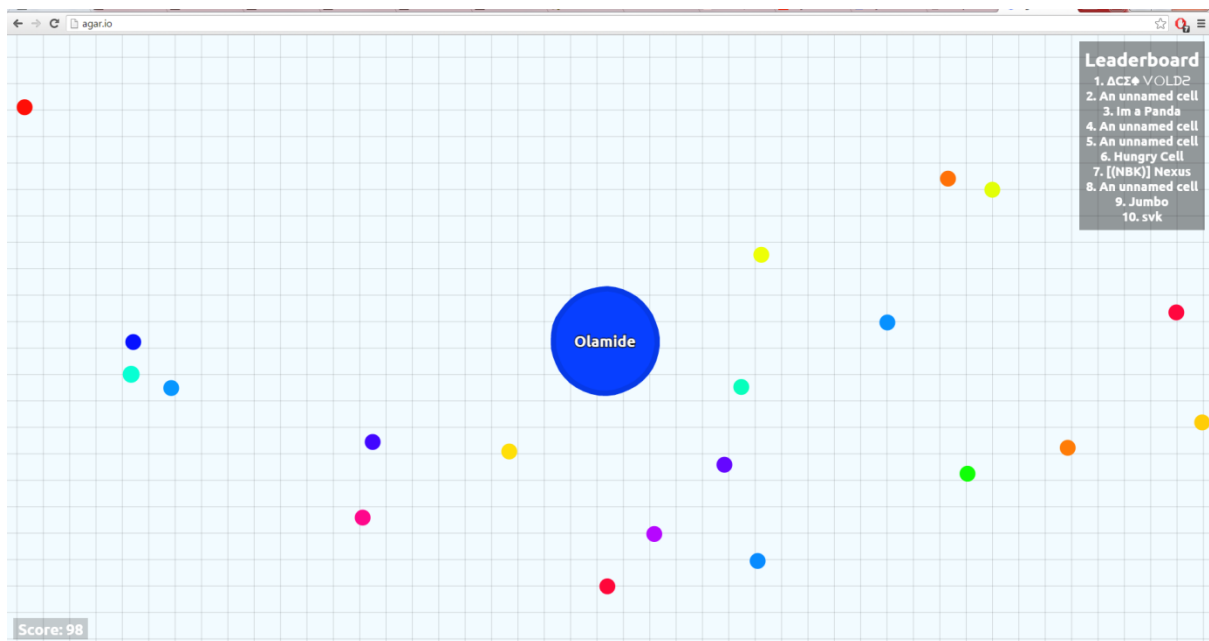
(ii) Investigation and Analysis

Analysis of Current System

There is no current system/version of my game as I will be trying to create a totally new concept. However, there is an online game very similar to the type of game I want to develop which I will use to analyse.

The game is called **agar.io**. You start off as a little circle and consume smaller, unmoving circles to grow. As you get bigger, you can consume other players and take their mass to get even bigger. When you are part of the ten biggest circles in the lobby you gain a place on the leader board. The game ends when you are consumed by another player.

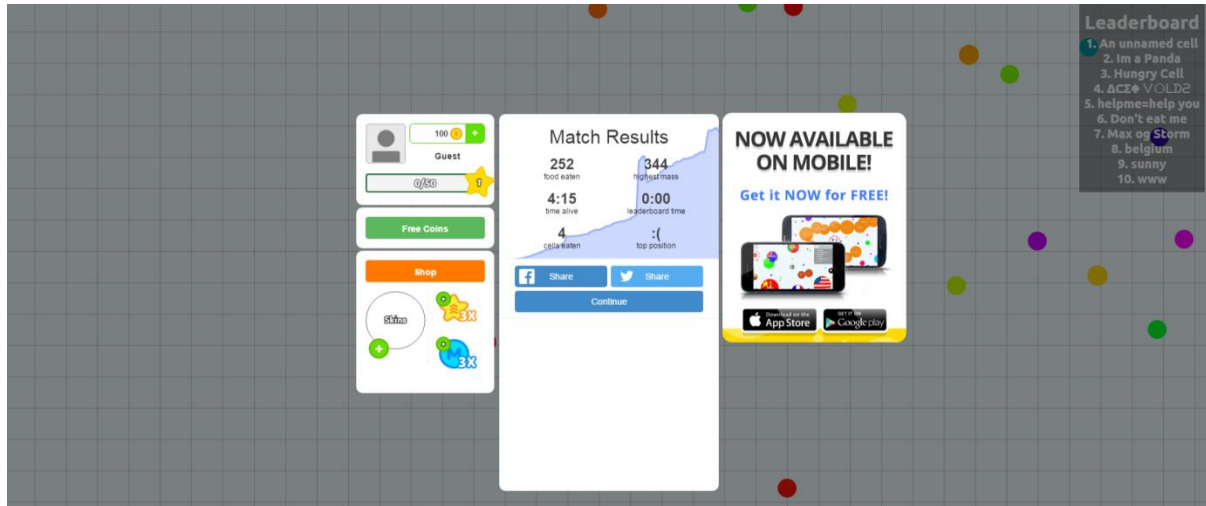
Here is a screen shot of me playing the game:



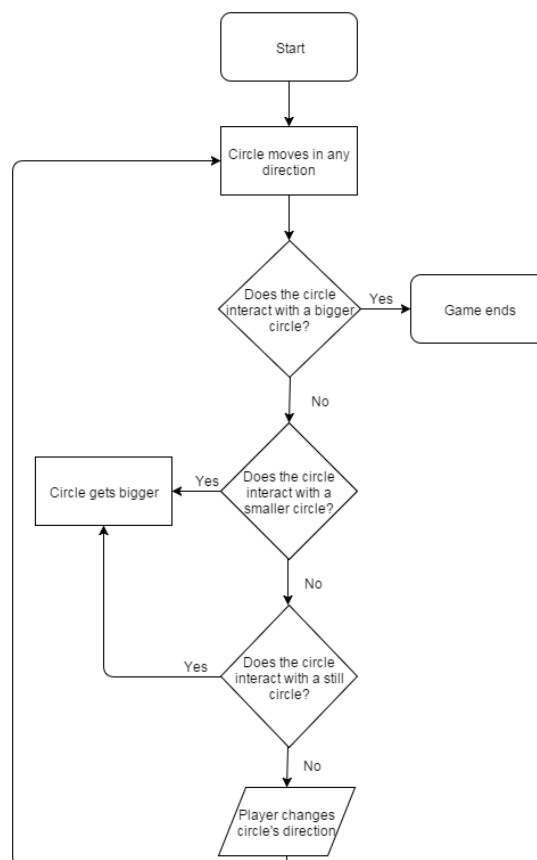
From playing this game I have spotted some features that are good and some features that are bad and I didn't like personally. For example, I noticed that if nothing is happening like there are no interactions with any other circles, the game starts to get boring. There should be something in the game to either stop it from going on for too long e.g. a time limit and/or something that adds incentive to take risks for the player to try and do as well as possible before the game ends. Special items are also usually included in games like this but this game has nothing. Also I experienced lag while playing the game. Lag is very annoying as it disrupts gameplay severely. As long as a game is online, there is no guarantee that there will be no lag as connections can be from/to anywhere. Also, the game is limiting with it being online. Players without an internet connection would not be able to play.

Here is a screen shot of what is outputted when the game has ended:

It includes the amount of still circles consumed, total game time, the amount of other players consumed, my highest size, the amount of time spent on the top ten leader board and if I achieved top position or not.

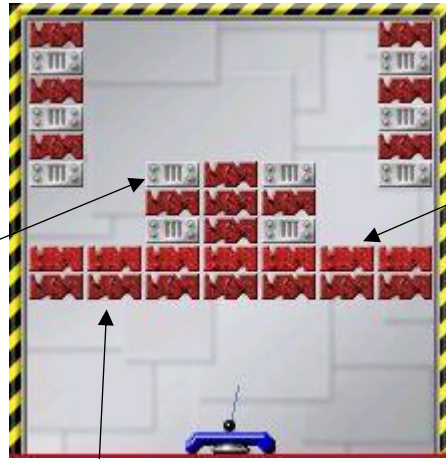


Here is a simple process model/system flowchart of how a typical game of **agar.io** is played:



To help me broaden my investigation, I will analyse another game which I feel is similar to the system I want to create but in different ways than the game I have already analysed. The new game I am going to analyse is **BrickBreaker**. The objective of the game is to try and get a score as high as possible by clearing all bricks on the screen with a miniature black ball which has a paddle base. There are 34 levels and each level gets harder with features like moving and unbreakable bricks. The ball hitting the paddle towards the middle keeps the ball moving at a slow speed but hitting towards the edge of the paddle increases the ball's speed to a fast speed. When all 34 levels have been completed, the ball's speed is set to permanently fast.

Metal brick: Unbreakable.
No amount of hits breaks
it.



Soft brick: Takes one hit
from the ball to break.

Hard brick: Takes two hits
from the ball to break.



The game has a power-up feature which I like and influenced my decision to include it as part of my analysis also. On collisions with the bricks, at a random rate capsules will fall with different power-ups that can be collected by the paddle. They are:

Life – Gives you an extra life (you start with three)

Multi – Puts four balls in play

Catch – Enables you to catch and hold the ball. After catching the ball, you can alter the angle at which the ball is shot

Laser – Gives you unlimited lasers that can damage regular (soft & hard) bricks. It takes two hits with a laser to damage a brick

Gun – Gives you three bullets that immediately destroy any brick. The gun is the only weapon that can destroy the unbreakable bricks

Long – Makes your paddle longer. This makes it easier to catch errant balls

Slow – Temporarily slows down the ball and halts the bricks from descending. After a certain number of bounces, the ball will switch to a fast speed, and the bricks will resume their descent

Bomb – Turns the ball into a bomb that destroys the next brick it comes into contact with, while also damaging adjacent bricks

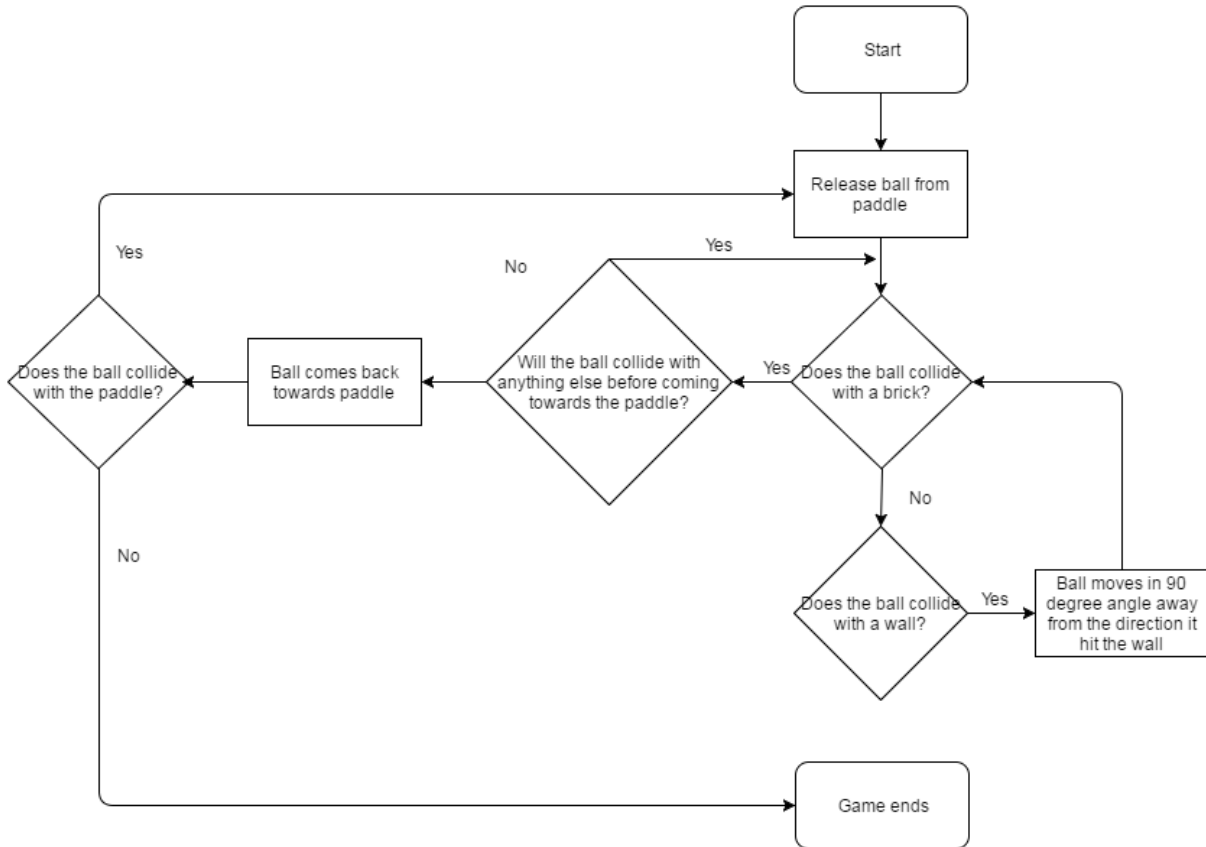
Wrap – Allows you to move the capsule beyond the edge of the screen

Flip – Changes the direction of the paddle

Similarities between **agar.io** and **BrickBreaker** are that the player controls a round shape object that collides with other items on the game screen. This, however, is where the similarities end. **BrickBreaker** collide with and bounces off walls and **agar.io** doesn't. And **BrickBreaker** has power-ups that **agar.io** simply does not. Lastly, **agar.io** is online and you interact with other players only and the size of the ball increases and decreases whereas **BrickBreaker** is the opposite to all of these conditions.

Here is a simple process model/system flowchart of how a typical game of **BrickBreaker** is played:

I was particularly intrigued with the loops caused by the two different collisions available to the ball



Requirements from Investigation and Analysis of Current System

- Something to stop the game getting boring
 - Could be a timer
 - Could be something to encourage the player to take risks
 - Could be a special item e.g. a power-up
- An offline game to remove the problem of lag and make the game more accessible

Investigation into New System

Having carried out a brief research and analysis of similar systems, I still feel there is more information for me to gather. Investigation is needed to find out user requirements, design requirements, requirements specification etc. This is so that the end user is fully satisfied with the final product and all requirements are met fully. I am going to use various methods of investigation such as questionnaires, observations and interviews.

I intend to question my brother and four of his classmates/friends as they are primary example of my end user target (they are all 8 and 9 years old) and will be the ones, ultimately, using the final product (the game).

I will then observe five of my classmates and interview one of them to gather up more requirements and get a more mature overview of the situation.

I then aim to conduct a second questionnaire on all users based on the first questionnaire and my interview. All ten people will be used as my end user group and will be returned to for feedback and improvement throughout the system development life cycle.

I am going to use two questionnaires. Questionnaires are useful because they are a quick and easy method of data collection e.g. the questionnaire can be made and handed out in one day and the results given back either the same day or the next day. Questionnaires are very practical and the results are easy to analyse. The first questionnaire will help me gather up initial end user views and requirements. The second will help me to refine these requirements and create the specification.

Questionnaire 1

This is my first questionnaire with the questions that will be asked and the justification for the questions, and then the answers/findings clearly outlined below:

Questions	Justification
Have you ever played pong or snake?	To get an idea of how many of my end users have played the games that my concept is based off.
Have you ever played agar.io or BrickBreaker?	To get an idea of how many of my end users have already played similar versions of the game I'm developing.
What do you like about the games?	To try and gauge the user requirements that are wanted.
What do you not like about the games?	To try and gauge the user requirements that are not wanted.
Do you prefer the online (agar.io) or the offline (pong, snake and BrickBreaker)?	To see what type of game is preferred.
Do you think that a game fusing pong and snake would be good?	To see if there is an actual desire for my game.

Answers:

Have you ever played pong or snake?

- Yes (5)*
- No (0)

Have you ever played agar.io or BrickBreaker?

- Yes (3)
- No (2)

What did you like about the games?

- Fun
- Easy to play
- Simple
- There is a point scoring aspect
- High intensity

What did you not like about the game?

- Gets boring sometimes
- Lag

Do you prefer the online (agar.io) or the offline (pong, snake and BrickBreaker)?

- Agar.io (4)
- Pong, snake and BrickBreaker (1)

Do you think that a game fusing pong and snake would be good?

- Yes (5)
- No (0)

* - The integer inside the parenthesis indicates the number of responses for that particular answer to the question asked

Analysis of Questionnaire 1

Everyone out of the five who completed the questionnaire had played pong or snake, so it was a good base to start with as they all had experience of playing that type of game. I knew I could gather dependable results.

Those who had played the three games I mentioned said they enjoyed them but I now realise that there is a real need for my system because the dislikes of the game matched up to most of my original assumptions. From the questionnaire, I see that I can create a game which eliminates the users' dislikes and expands on and improves their likes. For example, for their likes, I can design a fun but simple game with a friendly user interface and even though it is a new concept, make the game very easy to pick up first time. However some things I simply cannot implement because of obvious restrictions like a leader board (agar.io) and the various complex power-ups (BrickBreaker). For their dislikes, I can add extra features like power-ups and go ahead with my initial idea of having a time where the game will stop to alleviate boredom.

Requirements from Questionnaire 1

- Easy to pick up if playing for the first time
- Point scoring aspect

Even though I want to develop the game mainly for younger students, my game is accessible and can be played by anyone so I also want to observe and interview my classmates to gather up more requirements, and get a more experienced and mature point of view on things. I plan to observe five of my classmates (who are 17 and 18 years old) playing agar.io and then conduct an in depth interview with only one of them.

I am going to use one interview. An interview is a good way of collecting data because it is a face to face conversation and the interviewer can acquire requirements first hand. The

interviewee may be able to express in words what they could not do in another form of data collection like a questionnaire.

Interview

This is my interview with the questions that will be asked and the justification for the questions clearly outlined below:

Questions	Justification
Have you played agar.io or BrickBreaker before?	To see if my interviewee has already played similar versions of the game I'm developing.
Did you enjoy the games?	To try and see if they are a good similarity.
Would an offline based game similar to agar.io meet your needs/requirements and what new features would you like to see?	To try and see if my game is actually requested and to gauge some initial requirements.
Would it be a good introduction of the game to primary school children?	To see if my end user choice is a good decision.

Answers:

Have you played agar.io or BrickBreaker before?

- Yes.

Did you enjoy the games?

- Yeah I did because they are quite new and fresh. Not like any of the old school games. Very interesting.

Would an offline based game similar to agar.io meet your needs/requirements and what new features would you like to see?

- Yes. I think it being offline would make it more accessible because I would be able to play it even without connection to the internet. The only new feature I want to see, well it isn't new, but, a smaller screen/canvas that the sprite moves in because the agar.io platform is too big. I can't see what's happening across the whole screen. Otherwise, I would like to see the regular features just more refined.

Would it be a good introduction of the game to primary school children?

- Yes. It gives them a new game concept without all the hassle of online play so definitely.

Analysis of interview

My in depth interviewee again confirmed the need for my game because he would like an offline game and thinks my end user target is a good decision; that it would be beneficial for primary school children. He agrees with my previous end users that online play isn't suitable enough and believes the concept is good and that helps me to better refine my requirements.

Requirements from Interview

- Offline is definitely the best way to develop my game as my interviewee also acknowledges the internet connectivity problem
- Make the gameplay screen small enough so that the player can see everything that is happening on screen

Observation

I am going to use the method of observation as I can see my end users using the similar system first hand. With this method of investigation I can make my own assumptions and gather my own findings and conclusions without the opinions and answers of my end users.

Analysis of observation

I observed that my classmates were quite skilled in playing agar.io and the ones who had never played before picked it up quite quickly. I assumed this was due to their age and experience in playing games. I could see that after a while, they started to move round aimlessly because of such a big playing canvas and not interacting with other circles. I know I can develop a game that doesn't have these inefficiencies.

Requirements from Observation

- Definitely make the gameplay screen small enough so that the player can see everything that is happening on screen

After analysing the results of my observations, questionnaire and interview, getting initial requirements from these methods of investigation, and considering which features that I can implement successfully, I have decided on the following requirements:

General System Requirements

- (1) My game will be an offline and in the form of an application
- (2) It will contain a child user friendly, easy to use GUI
- (3) There will be a circle sprite that the player will control
- (4) Consuming one type of object will make the circle increase in size
- (5) Consuming another type of object will make the circle decrease in size
- (6) There will be power-ups to add a different dimension in the game (e.g. a special object to make the circle grow by a bigger amount than the normal growth object)
- (7) The game will end when the circle has reached a certain size, whether an increase or decrease
- (8) There will be a timer to output elapsed game time and end the game if the game has not ended already with gameplay by a certain time
- (9) Colours determined by the end user group will be used

To determine whether my user group is satisfied with the features I have listed and to decide on more specific requirements and details such as colour schemes I will conduct a second, more comprehensive and rigorous interview to gauge a full insight in what should be developed and to get the full scope. This time it will be with my five primary user group and five classmates (secondary user group).

Questionnaire 2

This is my second questionnaire with the questions that will be asked clearly outlined below:

Questions:

- Are you happy with the requirements I have listed so far?
- If not, what changes should be made?
- How large should the screen size of the game be, in line with the monitor screen resolution of 1280 x 1024 pixels?
 - 700 x 700 pixels (large)
 - 500 x 500 pixels (medium)
 - 300 x 300 pixels (small)
- How big should the initial circle be?
 - 15 x 15 pixels (large)
 - 10 x 10 pixels (medium)
 - 5 x 5 pixels (small)
- How should the player control the circle?
- Should the circle start to move immediately or with the player's input?
- Should the circle be able to go through the borders of the game screen?
- When should the timer start?

- Should the timer be on screen at all times or just displayed after the game?
- What should the colour scheme and background colour be?
- How much should a good item increase points by?
 - 50
 - 100
 - 200
- How much should a bad item decrease points by?
 - 10
 - 25
 - 50
- How much should a power-up item increase points by?
 - 100
 - 150
 - 250

Answers:

Are you happy with the requirements I have listed so far?

- Yes (9)
- No (1)

If not, what changes should be made?

- The sprite should be a different shape.

How large should the screen size of the game be, in line with the monitor screen resolution of 1280 x 1024 pixels?

- 700 x 700 pixels (0)
- 500 x 500 pixels (1)
- 300 x 300 pixels (8)

How big should the initial circle be?

- 15 x 15 pixels (2)
- 10 x 10 pixels (6)
- 5 x 5 pixels (2)

How should the player control the circle?

- Arrow keys (3)
- Mouse movement (3)
- Mouse clicks (4)

Should the circle start to move immediately or with the player's input?

- Immediately (2)
- Player input (8)

Should the circle be able to go through the borders of the game screen?

- Yes (3)
- No(7)

When should the timer start?

- Immediately (5)
- After input (5)

Should the timer be on screen at all times or just displayed after the game?

- All times (9)
- After game (1)

What should the colour scheme and background colour be?

- Classic (6)
- Modern (4)

How much should a good item increase points by?

- 50 (4)
- 100 (4)
- 200 (2)

How much should a bad item decrease points by?

- 10 (1)
- 25 (3)
- 50 (6)

How much should a power-up item increase points by?

- 100 (0)
- 150 (8)
- 250 (2)

Analysis of questionnaire 2

The majority of my user group were happy with my original requirements so I can safely ignore the one objection and treat it as an anomaly. I feel like any other shape than a circle would be considerably more difficult to program. I now have most, if not all of the scope of the problem required to create a solution. Besides the features where a majority could not be reached, for example when the timer should be start and how much a good item should increase points in which votes were split amongst the group, I will go with the preferences of my end user group questionnaire to create a requirements specification.

Requirements from Questionnaire 2

- All of the results from this questionnaire are viewed as requirements and will go into the requirements specification outlined below.

Requirements Specification

Design Requirements

- o The screen size will be 300 x 300 pixels
- o The colour of the background will be Black
- o The colour of all text will be White
- o The colour of the buttons will be Black
- o The player controlled sprite will be a white circle 10 x 10 pixels
- o The colour of the circle will be White
- o The good item will be a square smaller than the circle
- o The colour of the good item will be blue
- o The bad item will be a square smaller than the circle
- o The colour of the bad item will be red
- o The power-up item will be a square smaller than the circle
- o The colour of the power-up item will be Yellow

Input Requirements

- o The player moves the circle by clicking on the screen where they want the circle to move to

Processing Requirements

- o It must be determined with each movement whether the circle interacts with a good or bad item, in which case the item will be randomly moved to another position on the game screen
- o If the circle interacts with a good item it should grow
- o Points should be increased by 100
- o If the circle grows to a certain size the game should end and a "You win! :D" message be displayed
- o If the circle interacts with a bad item it should shrink
- o Points should be decreased by 50
- o If the circle shrinks to a certain size the game should end and a "You lose! :(" message be displayed
- o The power-up item will appear at a certain time and position on the screen during gameplay
- o It must be determined with each movement whether the circle interacts with a power-up item, in which case the item will disappear and appear at another certain time and position on the game screen
- o If the circle interacts with a power-up item it should grow by more than the good item
- o Points should increase by 150
- o The timer should start as soon as play is pressed
- o System must increment counter for the in game timer for every second that has elapsed
- o It should not end until the game is ended by the circle getting too big or too small
- o If it reaches one minute the game should end and a "Time's up!" message should be displayed

Output Requirements

- o Current time to show the length of time played should be displayed at all times on screen
- o Current points should be displayed at all times on screen
- o The position of the circle will be displayed in real time
- o The position of all items will be displayed

Hardware & Software Requirements

I have decided to write the game using the latest version of Python for Windows (Python 3.5.0*). I chose Python because the code is very friendly and minimises and compresses lines so it's easy to follow. Also the IDLE environment and GUI's are very accessible. The hardware and software should be of minimum standard required to run this version of Python.

Hardware Requirements	Justification
PC with 1 GHz processor minimum (2.5 GHz processor minimum recommended)	Processing power required to run Python 3.5.0
Approximately 32 MB of RAM	RAM needed to run Python 3.5.0
Approximately 25 MB of hard disk space	Needed to store Python and all relevant files plus the game so they can be run from the hard drive
VGA monitor or higher	Needed to display and output gameplay
Mouse compatible with the PC	Needed to navigate menus (e.g. clicking play to play the game and instructions to view instructions)
Keyboard compatible with the PC	Needed for the player to choose and input so that they can make an attempt to guess a letter

Software Requirements	Justification
Microsoft Windows Vista or later	Minimum Operating System recommended to run 3.5.0
TKinter**	Python's standard Graphical User Interface, needed to allow the game to be played, the sprite and the objects be shown (object-oriented programming) and the player to interact unlike the Command Line Interface

* - There are alternative programming languages I could have used for the development of my new system. **C/C++ and C#** are very powerful languages which have Object Oriented Programming and are good for large scale desktop games. The downsides are that the languages are hard to program in and hard to make cross platform. **VisualBasic (VB)** is easy to use and pick up but does not feature Object Oriented Programming and the programming is very limited and restricted. **Java** and **JavaScript** are strong alternatives. They feature

Object Oriented Programming and are easy to debug bugs in. The only real downsides are: they use more memory than Python, I don't have much experience programming in them and they are used in web games that require Flash, however, my game is an offline based game.

** - There are alternative modules, GUI toolkits, programmes and applications I could have used for the development of my new system. **Pygame** is a very popular cross-platform set of Python module used to make games. The problem with it is that it is not bundled with Python so players of my game will have to download it separately; also personally I prefer the programming structure of Tkinter. **PyQT** is a cross-platform GUI toolkit that is implemented as a Python plug-in. It has many more features and widgets than TKinter and the end result (interfaces) will look better. The downsides, however, are just like Pygame in the sense that it will need to be downloaded and the code is harder to pick up and follow.

User Review

All ten of my users have read, analysed and agreed to the above requirements.

Ayo

Ayo

Luca

Luca

James

James

Esther

Esther

Aisha

Aisha

Jerin

Jerin

Ian

Ian

Nirmal

N. Singh

Hamzah

Hamzah

Danyal

D. Patel

*(b) Design***(i) Nature of the solution****Design objectives**

The design objectives will be of a similar nature to the design requirements in the requirements specification similar to the requirements given previously. For example, in my designs, I will be using all the colours listed in the design requirements. Designs are also used to show the design considerations, for example the points and timer will be displayed on the game screen at all times. This is to show the users how long they have left and their current points

To demonstrate this to my users, and get their feedback, I'm going to design the interfaces for each screen and then display them to my users one by one and then conduct more interviews to see if any improvements can be made.

Here are the design objectives:

Design Requirements

- o The screen size will be 300 x 300 pixels
- o The colour of the background will be Black
- o The colour of all text will be White
- o The colour of the buttons will be Black
- o The player controlled sprite will be a white circle 10 x 10 pixels
- o The colour of the circle will be White
- o The good item will be a square smaller than the circle
- o The colour of the good item will be blue
- o The bad item will be a square smaller than the circle
- o The colour of the bad item will be red
- o The power-up item will be a square smaller than the circle
- o The colour of the power-up item will be Yellow

Input Requirements

- o The player moves the circle by clicking on the screen where they want the circle to move to

Processing Requirements

- o It must be determined with each movement whether the circle interacts with a good or bad item, in which case the item will be randomly moved to another position on the game screen
- o If the circle interacts with a good item it should grow
- o Points should be increased by 100
- o If the circle grows to a certain size the game should end and a "You win! :D" message be displayed
- o If the circle interacts with a bad item it should shrink
- o Points should be decreased by 50

- o If the circle shrinks to a certain size the game should end and a “You lose! :(” message be displayed
- o The power-up item will appear at a certain time and position on the screen during gameplay
- o It must be determined with each movement whether the circle interacts with a power-up item, in which case the item will disappear and appear at another certain time and position on the game screen
- o If the circle interacts with a power-up item it should grow by more than the good item
- o Points should increase by 150
- o The timer should start as soon as play is pressed
- o System must increment counter for the in game timer for every second that has elapsed
- o It should not end until the game is ended by the circle getting too big or too small
- o If it reaches one minute the game should end and a “Time’s up!” message should be displayed

Output Requirements

- o Current time to show the length of time played should be displayed at all times on screen
- o Current points should be displayed at all times on screen
- o The position of the circle will be displayed in real time
- o The position of all items will be displayed

User Review

All ten of my users have read, analysed and agreed to the above requirements.

Ayo



Luca



James



Esther



Aisha



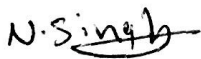
Jerin



Ian



Nirmal



Hamzah

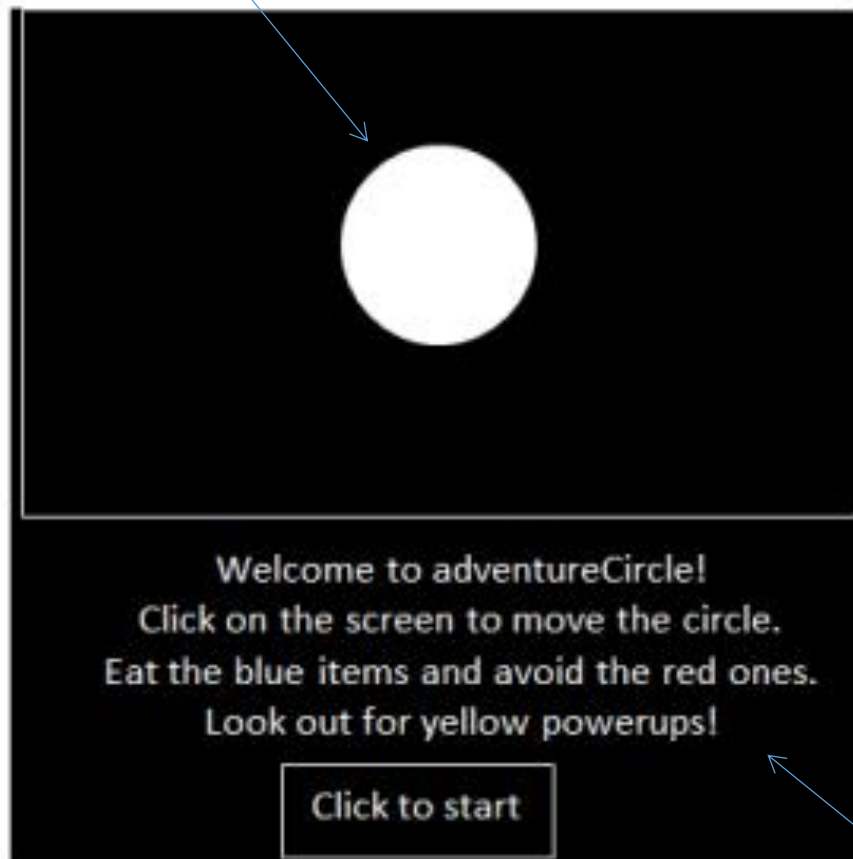


Danyal



Main Menu Screen

The circle sprite, which is controlled by the player.



The "Click to start" button. When clicked the timer starts and the game begins.

The title of the game and the instructions.

Interface Design: Main Menu Screen

Design	Property	Description
Background	Colour	Black RGB – (0, 0, 0)
	Size	300 x 300 pixels
Title	Text	Welcome to adventureCircle!
	Font	Calibri, Size 11
	Colour	White RGB – (255, 255, 255)
	Alignment	Centre
Instructions	Text	Click on the screen to move the circle. Eat the blue items and avoid the red ones. Look out for yellow power-ups!
	Font	Calibri, Size 11
	Colour	White RGB – (255, 255, 255)
	Alignment	Centre
Button 1	Text	Click to start
	Font	Calibri, Size 11
	Colour	White RGB – (255, 255, 255)
	Alignment	Centre
Diagram	Object	Circle sprite
	Colour	White RGB – (255, 255, 255)
	Size	70 x 70 pixels

The initial design sizes are only for design purposes only and do not necessarily reflect the size of what the end result game will look like.

I will show this design to my end users and get their feedback by conducting an interview to see if they would make any changes.

Main Menu Screen Interview

Questions:

- What changes should be made to how the main menu screen looks, if any?
- What changes should be made to the functionality of the main menu screen, if any?

Answers:

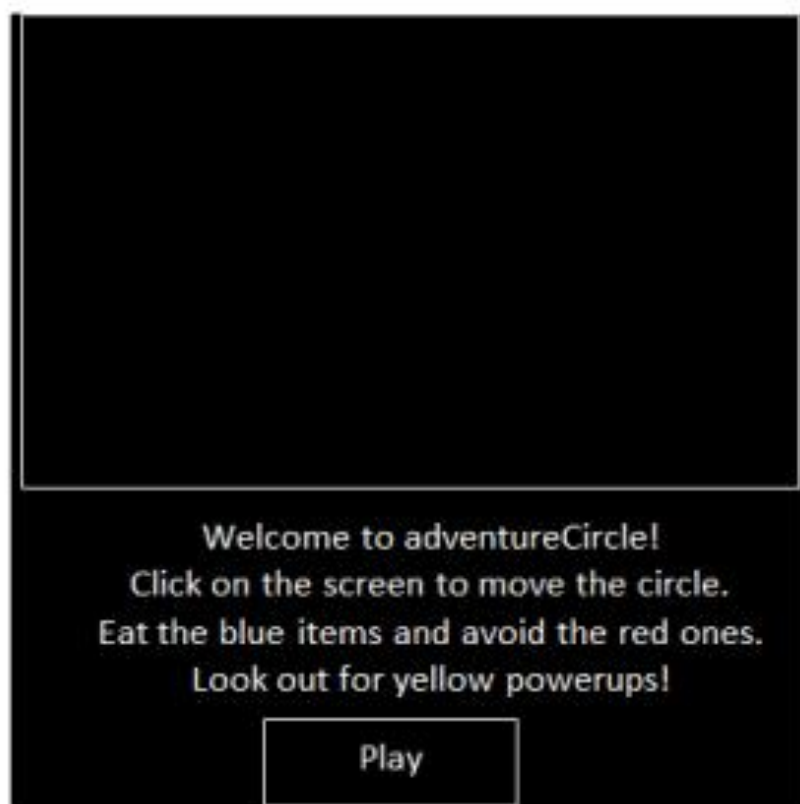
What changes should be made to how the main menu screen looks, if any?

- Remove the circle sprite
- Change "Click to start" to "Play"

What changes should be made to the functionality of the main menu screen, if any?

- None

Improved Main Menu Screen:



Gameplay Screen

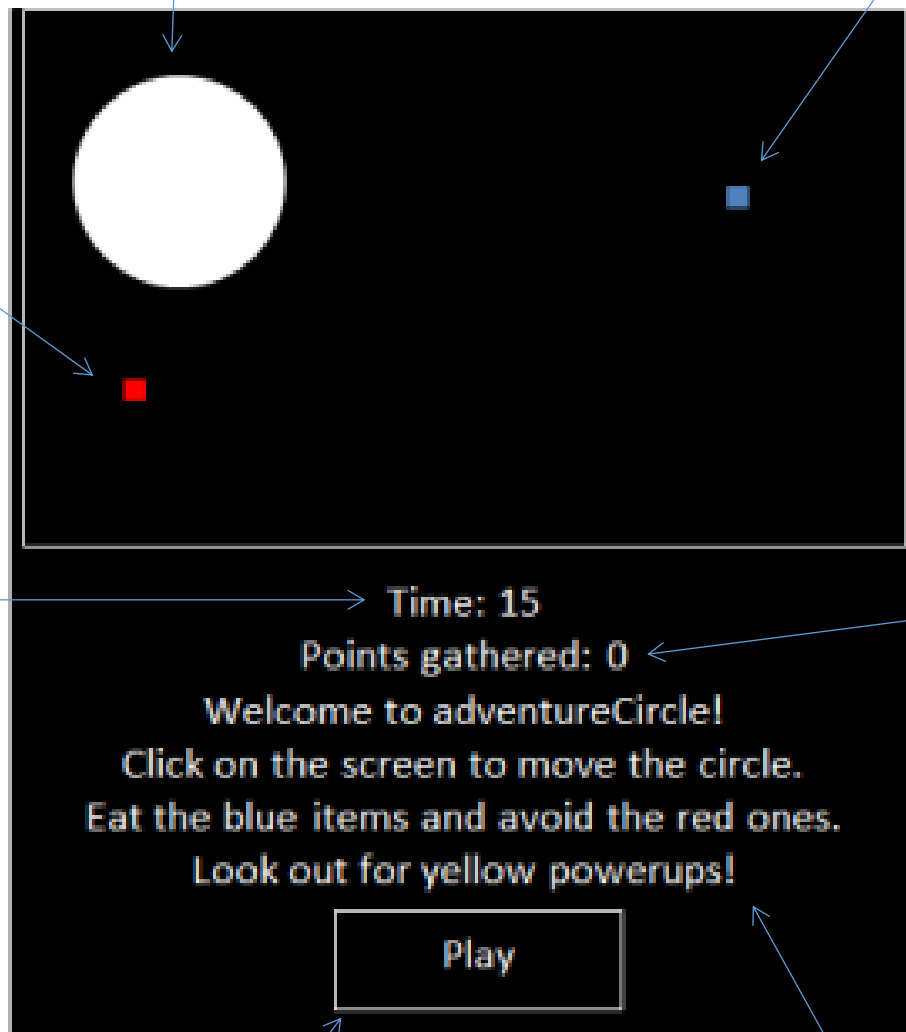
The circle sprite, which is controlled by the player.

The good item. When eaten the circle sprite grows.

The bad item. When eaten the circle sprite shrinks.

Text with the current time from the timer in the game. Includes [variable containing seconds].

Text with the amount of points gathered.



The "Play" button. When clicked the timer starts and the game begins.

The title of the game and the instructions.

Interface Design: Gameplay Screen

Design	Property	Description
Background	Colour	Black RGB – (0, 0, 0)
	Size (half of actual game size)	400 x 400 pixels
Time	Text	Second
	Font	Calibri, Size 11
	Colour	White RGB – (255, 255, 255)
Points gathered	Alignment	Centre
	Text	[Integer with amount of points gathered]
	Font	Calibri, Size 11
	Colour	White RGB – (255, 255, 255)
	Alignment	Centre
Title	Text	Welcome to adventureCircle!
	Font	Calibri, Size 11
	Colour	White RGB – (255, 255, 255)
Instructions	Alignment	Centre
	Text	Click on the screen to move the circle. Eat the blue items and avoid the red ones. Look out for yellow power-ups!
	Object	Circle sprite
Diagram 1	Colour	White RGB – (255, 255, 255)
	Size	70 x 70 pixels
	Object	Good item
Diagram 2	Colour	Blue RGB – (79, 129, 189)
	Size	8 x 8 pixels
	Object	Bad item
Diagram 3	Colour	Red RGB – (255, 0, 0)
	Size	8 x 8 pixels

The initial design sizes are only for design purposes only and do not necessarily reflect the size of what the end result game will look like.

I will show this design to my end users and get their feedback by conducting an interview to see if they would make any changes.

Gameplay Screen Interview

Questions:

- What changes should be made to how the gameplay screen looks, if any?
- What changes should be made to the functionality of the gameplay screen, if any?

Answers:

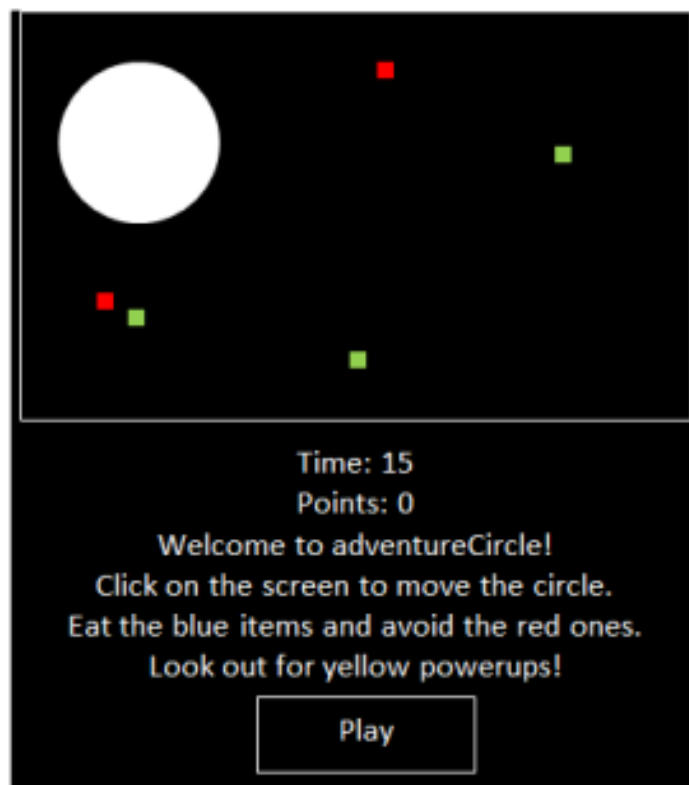
What changes should be made to how the gameplay screen looks, if any?

- Change the colour of the good item to Green
- Add more good and bad items, not just one of each
- Change "Points gathered:" to "Points"

What changes should be made to the functionality of the main menu screen, if any?

- A power-up item should flash to signify it is a power-up

Improved Gameplay Screen:



Changes to the design objectives

From the results of the interviews of the initial designs of the screens I have made amendments to the original design objectives. The removed requirements are highlighted red and the new requirements sections are made bold.

Design objectives:

Design Requirements

- o The screen size will be 300 x 300 pixels
- o The colour of the background will be Black
- o The colour of all text will be White
- o The colour of the buttons will be Black
- o The player controlled sprite will be a white circle **10 x 10 pixels**
- o **Most likely bigger**
- o The colour of the circle will be White
- o The good item will be a square smaller than the circle
- o The colour of the good item will be **blue**
- o **The colour of the good item will be green**
- o The bad item will be a square smaller than the circle
- o The colour of the bad item will be red
- o The power-up item will be a square smaller than the circle
- o The colour of the power-up item will be Yellow
- o **The power-up item will flash when active to signify it is a power-up**

Input Requirements

- o The player moves the circle by clicking on the screen where they want the circle to move to

Processing Requirements

- o It must be determined with each movement whether the circle interacts with a good or bad item, in which case the item will be randomly moved to another position on the game screen
- o If the circle interacts with a good item it should grow
- o Points should be increased by 100
- o If the circle grows to a certain size the game should end and a "You win! :D" message be displayed
- o If the circle interacts with a bad item it should shrink
- o Points should be decreased by 50
- o If the circle shrinks to a certain size the game should end and a "You lose! :(" message be displayed
- o **There should be one good and bad item on screen to start with but more than one of each item can be on the screen at any time as the game is played**

- o The power-up item will appear at a certain time and position on the screen during gameplay
- o It must be determined with each movement whether the circle interacts with a power-up item, in which case the item will disappear and appear at another certain time and position on the game screen
- o If the circle interacts with a power-up item it should grow by more than the good item
- o Points should increase by 150
- o The timer should start as soon as play is pressed
- o System must increment counter for the in game timer for every second that has elapsed
- o It should not end until the game is ended by the circle getting too big or too small
- o If it reaches one minute the game should end and a “Time’s up!” message should be displayed

Output Requirements

- o Current time to show the length of time played should be displayed at all times on screen
- o Current points should be displayed at all times on screen
- o The position of the circle will be displayed in real time
- o The position of all items will be displayed

User Review

All ten of my users have read, analysed and agreed to the above requirements.

Ayo



Luca



James



Esther



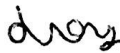
Aisha



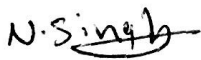
Jerin



Ian



Nirmal



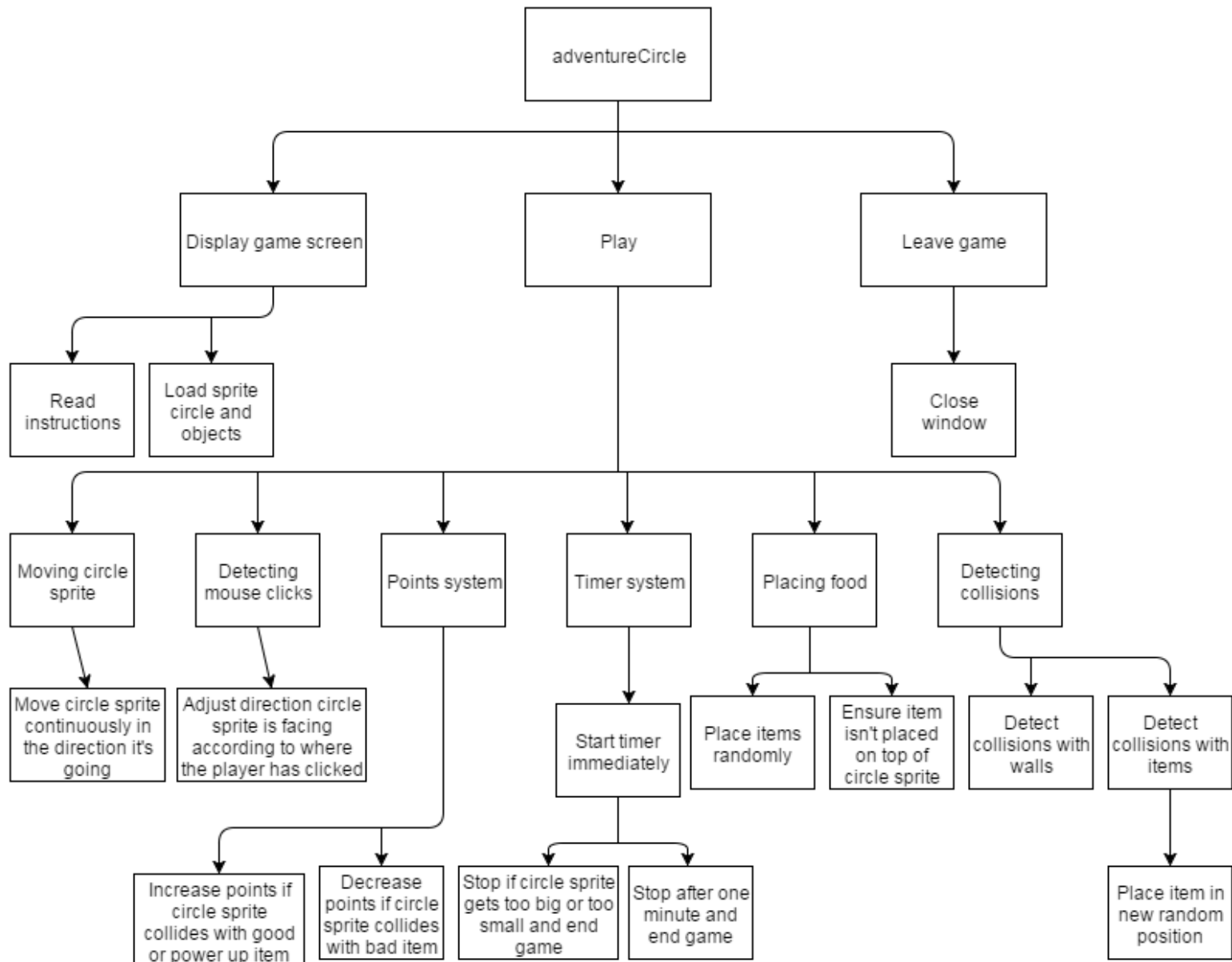
Hamzah



Danyal



Modular Design



Structure Diagram (Modular Design) Explained

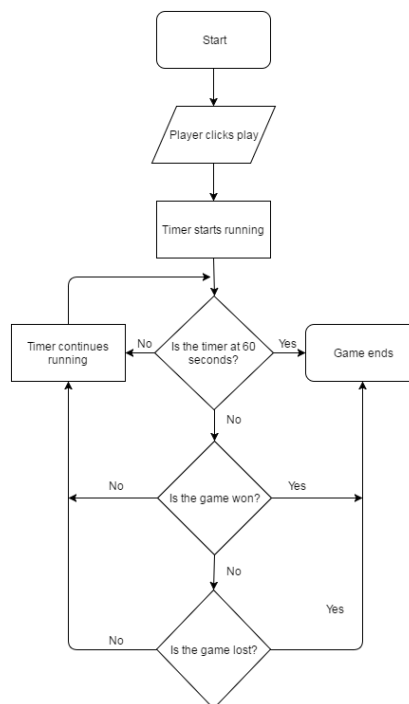
Display Game Screen

- Read instructions – The player has the option to read the instructions that are clearly laid out on the main menu screen
- Load sprite circle and objects – Displaying the game screen also loads all the things that will be on the gameplay screen

Play

- Moving circle sprite – When the circle sprite is in motion (has started moving) it continues moving in its current direction till it comes across a wall
- Detecting mouse clicks – The circle sprite adjusts its direction instantaneously (and in real time) in retaliation to player input which is a click of the left mouse button. Whichever of the eight programmed directions is where the circle sprite should now be travelling
- Points system – The points tally below the gameplay screen will increase by 100 points when the circle sprite consumes a good item (green) and decrease by 50 points when it consumes a bad item (red). If points is on 0 then minus numbers are allowed
- Timer system – The timer will start immediately play is clicked, regardless of whether the player has begun input (clicking on the screen) or not. The timer will stop when the game is won or lost and is ended e.g. if the circle sprite has reached maximum size and is too big or minimum size and is too small. Also the game will end and so will the timer automatically when one minute is reached

Here is a simple process model/system flowchart of how the timer system will work in the game:



- Placing food – All items (meaning good, bad and power-up) are placed randomly on the game canvas and ensured they aren't placed on top of the circle sprite
- Detecting collisions – The circle sprite detects and reacts to collisions with walls which move the sprite in a 90 degree angle away from the direction it hit the wall in. It also moves/places a new item in a random position on the game canvas

Leave Game

- Close window – The player is able to click the red X in the top right position of the game window the close and exit the game

Data Structure Design

My game will use many variables which are contained within the many procedures. They are all held under the one class “adventureCircle”, the title of the game.

None of my subroutines are functions, they are all procedures as none of them return just one value and they all use local variables (variables only usable in that part of the program).

Here are the variables:

Variable Name (Procedure Name/Names)	Data Type	Size (possible values)	Description	Sample Values	Validation
RUN __init__(self) start(self) run(self) end(self)	Boolean	FALSE – TRUE	Used to check whether to run the actual game or just set up the variables below it	TRUE	N/A
Canvas __init__(self) start(self) end(self) CreateGoodItem(self,ball) CreateBadItem(self,ball) CreatePowerUpItem(self,ball) paint(self)	Integer	Width = 0 – 300 Height = 0 – 300	The size in pixels of the whole game interface	Width = 144 Height = 251	N/A
TimerText __init__(self) run(self) paint(self)	String	N/A	Variable to add the label for the timer	N/A	N/A
PointsText __init__(self) run(self)	String	N/A	Variable to add the label for the points	N/A	N/A
Title __init__(self)	String	N/A	Variable to add the label for the title and welcome message	N/A	N/A
Instructions __init__(self)	String	N/A	Variable to add the label for the instructions	N/A	N/A
Timer start(self) run(self) CreateGoodItem(self,ball)	Integer	0 – 60	The timer that determines when the game starts and ends	7	If the time <0 or >480 then return error as the game either should

CreateBadItem(self, ball) CreatePowerUpItem(self, ball) paint(self)					not have started or have ended and the time should always stay within the range
GoodItemX start(self) CreateGoodItem(self, ball)	Random Integer	50 – 250	Holds a random x co-ordinate of a good item	21	If $x < 50$ or $x > 250$ then return error as the item would be out of the range
GoodItemY start(self) CreateGoodItem(self, ball)	Random Integer	50 – 250	Holds a random y co-ordinate of a good item	69	If $y < 50$ or $y > 250$ then return error as the item would be out of the range
BadItemX (start(self) CreateBadItem(self, ball)	Random Integer	50 – 250	Holds a random x co-ordinate of a bad item	9	If $x < 50$ or $x > 250$ then return error as the item would be out of the range
BadItemY start(self) CreateBadItem(self, ball)	Random Integer	50 – 250	Holds a random x co-ordinate of a bad item	77	If $y < 50$ or $y > 250$ then return error as the item would be out of the range
PowerUpItemX start(self) CreatePowerUpItem(self, ball)	Random Integer	50 – 250	Holds a random x co-ordinate of a power-up item	201	If $x < 50$ or $x > 250$ then return error as the item would be out of the range
PowerUpItemY start(self) CreatePowerUpItem(self, ball)	Random Integer	50 – 250	Holds a random x co-ordinate of a power-up item	115	If $y < 50$ or $y > 250$ then return error as the item would be out of the range
Points (start(self)) run(self) CreateGoodItem(self, ball) CreateBadItem(self,	Integer	$-\infty - \infty$	The amount of points the player has amassed while playing	1470	N/A

ball) CreatePowerUpItem (self,ball)					
x start(self) paint(self) MoveCircleSprite(se lf, b,speed) MouseClicks(self,ev ent)	Integer	0 – 100	Contains the x position of the circle sprite, ranges between 0 and 100 because of the game screen size	13	If x<0 or x>100 then return error as circle sprite would be off the screen
y start(self) paint(self) MoveCircleSprite(se lf, b,speed) MouseClicks(self,ev ent)	Integer	0 – 100	Contains the y position the character , ranges between 0 and 100 because of the game screen size	88	If y<0 or y>100 then return error as circle sprite would be off the screen
tempx start(self) MouseClicks(self,ev ent)	Integer	0 – 100	Used to determine the x position of the circle sprite in terms of the game screen to adjust for wall collisions	51	If x<0 or x>100 then return error as circle sprite would have gone past the wall
tempy start(self) MouseClicks(self,ev ent)	Integer	0 – 100	Used to determine the x position of the circle sprite in terms of the game screen to adjust for wall collisions	2	If y<0 or y>100 then return error as circle sprite would have gone past the wall
UP start(self) MoveCircleSprite(se lf, b,speed) MouseClicks(self,ev ent)	Boolean	FALSE – TRUE	Used to check whether the circle sprite is moving in an upwards direction including up-left and up-right	TRUE	N/A
DOWN start(self) MoveCircleSprite(se lf, b,speed) MouseClicks(self,ev ent)	Boolean	FALSE – TRUE	Used to check whether the circle sprite is moving in an downward direction including down-left and down-right	FALSE	N/A

LEFT start(self) MoveCircleSprite(self, b, speed) MouseClicks(self, event)	Boolean	FALSE – TRUE	Used to check whether the circle sprite is moving in a left direction including up-left and down-left	TRUE	N/A
RIGHT start(self) MoveCircleSprite(self, b, speed) MouseClicks(self, event)	Boolean	FALSE – TRUE	Used to check whether the circle sprite is moving in a right direction including up-right and down-right	FALSE	N/A
Size start(self) run(self) CreateGoodItem(self, ball) CreateBadItem(self, ball) CreatePowerUpItem(self, ball) paint(self)	Float	$0 - \infty$	Sets the initial size and growth and shrinking of the circle sprite	3.5	N/A

(ii) **Algorithms**

My program will require algorithms written in many modules (I have listed them in the variables table above as procedure names). I will name each module in plain English with each algorithm bulleted below. I will then describe each algorithm so it is easier to understand using pseudo code.

1. Game start up
 - Create interface
 - Load initial sprite and items
2. Place items
 - Place new items randomly on the canvas and make sure they're not placed on top of each other or the circle sprite
3. Detect mouse clicks
 - Make sure the circle sprite changes to the correct direction when a position on the canvas is clicked and that the sprite does not change direction when it is clicked itself
4. Circle sprite
 - Movement and wall collisions
5. Game end
 - Stop the program

1. Game start up

Create interface

Written in pseudo code. This algorithm describes using Tkinter to make the game interface, place all the buttons and labels and lay the canvas for the actual game. From the **__init__(self)** procedure.

Create new game loop

Create new frame

Set background colour of frame to black

Create new canvas

Set background colour of canvas to black

Set width of canvas to 300

Set height of canvas to 300

Draw title label

Draw instructions label

Draw play button

Load initial sprite and items

Written in pseudocode. This will load the circle sprite and initial good and bad items. The algorithm for the circle sprite first makes sure the canvas (game screen) is blank. I will just show the algorithm for the good item as it is the same as the bad item. From the **paint(self), CreateGoodItem(self,ball)** procedures.

DELETE all in canvas

IF (timer DIV 100) <= 60 THEN

IF ball size > 0 THEN

Create ball

Fill white

IF LEN x co-ordinate of good item < (timer DIV 1500 + 1) THEN

IF LEN y co-ordinate of good item < (timer DIV 1500 + 1) THEN

Create good item in canvas co-ordinates [50 – 250]

2. Place items

Place new items randomly on the canvas and make sure they're not placed on top of each other or on the circle sprite

This algorithm detects when the circle sprite collides with an item and so places a new item in a random position. I will just show the algorithm for the good item as it is the same as the bad item. From the **CreateGoodItem(self,ball)** procedure.

1. FOR INT i = 0 to x co-ordinate of good item
2. IF LEN [overlapping] of good item IS NOT 1 THEN
3. IF ball in canvas [overlapping]
4. Points + 100
5. Circle sprite size + 0.5
6. New x co-ordinate for good item
7. New y co-ordinate for good item

I am going to perform a dry run where I will test the algorithm to see if it works successfully. I will do one test to make sure that the items are placed randomly. I will run through one full test in which one new good item is placed on the game screen.

Line	i	x	Y	Points	Size	Output	Comment
1	0						
2		159	221	0	3		Good item has been given a random x and y position
3				0	3		TRUE, I moved the circle sprite around till it hit a good item
4				100	3	Points + 100	
5				100	3.5	Size + 0.5	
6		118		100	3.5	New x co-ordinate	Good item given a new random x co-ordinate
7			98	100	3.5	New y co-ordinate	Good item given a new random y co-ordinate

3. Detect mouse clicks

Make sure the circle sprite changes to the correct direction when a position on the canvas is clicked and that the sprite does not change direction when it is clicked itself

Written in pseudocode. This algorithm detects when a position on the game screen that is not the circle sprite itself is clicked and changes direction accordingly. From the **MouseClicks(self,event)** procedure.

```

tempx = eventx
tempy = eventy
IF eventx > selfx AND selfx IS NOT tempx THEN
    RIGHT = TRUE
    LEFT = FALSE
ELSEIF eventx < selfx AND selfx IS NOT tempx THEN
    LEFT = TRUE
    RIGHT = FALSE
ELSE THEN
    selfx = tempx
    RIGHT = FALSE
    LEFT = FALSE
IF eventy > selfy AND selfy IS NOT tempy THEN
    DOWN = TRUE
    UP = FALSE
ELSEIF eventy < selfy AND selfy IS NOT tempy THEN
    UP = TRUE
    DOWN = FALSE
ELSE THEN
    selfy = tempy
    DOWN = FALSE
    UP = FALSE

```

4. Circle sprite

Movement and wall collisions

This algorithm tracks the direction and movement of the circle sprite and changes direction to the opposite direction when it hits a wall. From the **MoveCircleSprite(self, b, speed)** procedure.

```

1. IF UP == TRUE AND selfy - b > 0 THEN
2.     selfy = speed
3.     ELSEIF UP == TRUE AND selfy - b <= 0 THEN
4.         UP = FALSE
5.         DOWN = TRUE
6.     IF DOWN == TRUE AND selfy + b < 300 THEN
7.         selfy = speed
8.     ELSEIF DOWN == TRUE AND selfy + b >= 300 THEN
9.         DOWN = FALSE
10.        UP = TRUE
11.    IF LEFT == TRUE AND selfx - b > 0 THEN
12.        selfx = speed
13.    ELSEIF LEFT == TRUE AND selfx - b <= 0 THEN
14.        LEFT = FALSE
15.        RIGHT = TRUE
16.    IF RIGHT == TRUE AND selfx + b < 300 THEN
17.        selfx = speed
18.    ELSEIF RIGHT == TRUE and selfx + b >= 300 THEN
19.        RIGHT = FALSE
20.        LEFT = TRUE

```

I am going to create a dry run table to test if the algorithm works. I will run the test until all four walls are collided with. I will start the test by moving the circle in an upwards direction. b represents the vicinity of the circle sprite so it still changes direction when it collides with a wall. b should never be < 0 or > 300 or else the program should return an error as that means either the circle sprite is bigger than the game screen/not on the game screen or the circle sprite has gone through the wall. I will aim to keep b at thirty by not consuming any items.

Line	selfx	selfy	b	UP	DOWN	LEFT	RIGHT	Output	Comment
1	75	70	30	TRUE	FALSE	FALSE	FALSE		The mouse is clicked above the circle sprite and it is moving in an upwards direction
2								TRUE, go to	

								line 3	
3	75	30	30	TRUE	FALSE	FALSE	FALSE		
4				FALSE					Circle sprite changes direction
5					TRUE				Circle sprite changes direction
6	75	125	30	FALSE	TRUE	FALSE	FALSE		Circle sprite moving in a downward direction
7								TRUE, go to line 8	
8	75	270	30	FALSE	TRUE	FALSE	FALSE		
9				TRUE					Circle sprite changes direction
10					FALSE				Circle sprite changes direction
11	74	261	30	FALSE	FALSE	TRUE	FALSE		The mouse is clicked to the left of the circle sprite and it is moving in an leftward direction
12								TRUE, go to line 13	
13	30	261	30	FALSE	FALSE	TRUE	FALSE		
14						FALSE			Circle sprite changes direction
15							TRUE		Circle sprite changes direction
16	125	261	30	FALSE	FALSE	FALSE	TRUE		
17								TRUE, go to line 18	
18	270	261	30	FALSE	FALSE	FALSE	TRUE		
19						TRUE			Circle sprite changes direction
20							FALSE		Circle sprite changes direction

5. Game end

End program

Written in pseudocode. The algorithm shows the three conditions leading to the game program being terminated, the player equivalent of 'Game Over'. The first is if the circle sprite gets too big, the second if it gets too small and the third if the game timer gets to sixty seconds. From the **paint(self)** procedure.

```
If 10 * ball size > 600 THEN
    PRINT "You Win! :D"
    END
ELSE PRINT "You Lose! :("
    END
ELSE PRINT "Time's up!"
    END
```

(iii) Test Strategy

To test my program (adventureCircle) before I present the game to my end users to ensure the game is of high quality, free of any bugs and meets all of the requirements in the specification. The test will follow these stages:

- ❖ Alpha stage – Black box testing will be used during and after the development to make sure the game runs as it should. The testing will be done by the software developer (me) and will be the last chance to spot bugs and fix them before releasing a version to end users to test. I'm going to stimulate end users by performing tasks that they will normally perform
- ❖ Beta stage – During this stage, I will get my primary end user to test the game to make sure he is happy with the game. Beta testing is a form of external user acceptance testing. It reduces product failure risks and provides increased quality of the product through customer validation. It is the final test before finishing the game completely. Direct feedback from customers is a major advantage of Beta testing. This testing helps to test the product in a real time environment.

During these tests, I will debug any bugs that start to occur and I will provide a solution to any problems that my end users have of the game.

When the game is finished, I will test it with black box testing using the test plan that I have created and then fix any errors and bugs that occur during this test.

A screenshot will be taken as evidence once it has been tested later for each test.

Test no.	What is being tested	Input method	Type	Input	Expected result
----------	----------------------	--------------	------	-------	-----------------

Tests involving movement of the circle sprite

1	Circle sprite movement Up movement of circle sprite using valid left mouse button click above circle sprite	Clicking on the game screen with the left mouse button away from the circle sprite	Normal	Click directly above the circle sprite	The circle sprite will begin to move in an upward direction or keep moving in an upward direction if already doing so
2	Circle sprite movement Down movement of circle sprite using valid left mouse button click below circle	Clicking on the game screen with the left mouse button away from the circle	Normal	Click directly below the circle sprite	The circle sprite will begin to move in a downward direction or keep moving in a

	sprite	sprite			downward direction if already doing so
3	Circle sprite movement Left movement of circle sprite using valid left mouse button click to the left of circle sprite	Clicking on the game screen with the left mouse button away from the circle sprite	Normal	Click directly to the left of the circle sprite	The circle sprite will begin to move in a leftward direction or keep moving in a leftward direction if already doing so
4	Circle sprite movement Right movement of circle sprite using valid left mouse button click to the right of circle sprite	Clicking on the game screen with the left mouse button away from the circle sprite	Normal	Click directly to the right of the circle sprite	The circle sprite will begin to move in a rightward direction or keep moving in a rightward direction if already doing so
5	Circle sprite movement No movement of circle sprite with no player input	No input	Normal	No click	The circle sprite should continue to move in whatever direction it is moving or keep still if not moving
6	Circle sprite movement No movement of circle sprite when edges of circle sprite clicked	Clicking on the edge of the circle sprite with the left mouse button away from the circle sprite	Extreme	Edges of the circle sprite	The circle sprite should continue to move in whatever direction it is moving or keep still if not moving
7	Circle sprite movement No movement of circle sprite when edges of	Clicking on the edge of the game screen with the left mouse	Extreme	Edges of the game screen	The circle sprite should continue to move in whatever direction it is

	game canvas (walls) are clicked	button away from the circle sprite			moving or keep still if not moving
8	Circle sprite movement No movement of circle sprite when right mouse button is clicked	Clicking on the game screen with the right mouse button away from the circle sprite	Abnormal	Right mouse button click	The circle sprite should continue to move in whatever direction it is moving or keep still if not moving. The right mouse button click is not registered to function
9	Circle sprite movement No movement of circle sprite when letter key is pressed (W)	Pressing the W key	Abnormal	W key	The circle sprite should continue to move in whatever direction it is moving or keep still if not moving. No key is registered to function

Tests involving game navigation

10	Display game Display interface on game canvas with valid left mouse button click on play button	Clicking the play button on the main menu	Normal	Left mouse button click	Display game
11	Display game Display interface on game canvas with long holding (ten seconds) left mouse button click on play button	Clicking the play button on the main menu	Extreme	Left mouse button click for an extended period of time (e.g. 10 seconds)	Display game
12	Display game	Clicking the	Extreme	Left mouse	Display game

	Display interface on game canvas with valid left mouse button click on edge of play button	edge of the play button on the main menu		button click at the edge of the play button	
13	Display game Display interface on game canvas with invalid right mouse button click on play button	Clicking the play button on the main menu with the right mouse button	Abnormal	Right mouse button click	Nothing happens

Tests involving points system

14	Points to end the game Consume six consecutive good items to acquire 600 points and end the game with a "You win" message	Consume 6 good items > starting circle sprite size (600 points)	Normal	Consume 6 good items consecutively	Game ends and "You win! :D" message displayed
15	Points to end the game Consume six consecutive bad items to acquire -300 points and end the game with a "You lose" message	Consume 6 bad items < starting sprite (-300 points)	Normal	Consume 6 bad items consecutively	Game ends and "You lose! :(" message displayed

Tests involving timer system

16	Timer functionality Immediate starting of the timer	Clicking the play button on the main menu	Normal	Left mouse button click	Timer should start immediately play is clicked
----	---	---	--------	-------------------------	--

17	Timer functionality Timer should count in seconds one second after the other	No input	Normal	No click	Timer should count in individual seconds and not any faster or slower
18	Timer functionality Timer should stop immediately 60 seconds is reached	No input	Normal	No click	Game ends and "You win! :D" or "You lose!" message displayed
19	Timer functionality Change timer functionality by clicking on it with the left mouse button	Clicking on the timer with the left mouse button	Abnormal	Left mouse button click	The timer should continue counting as normal

User review

All ten of my users have read, analysed and agreed to the above test strategy.

Ayo



Luca



James



Esther



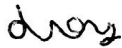
Aisha



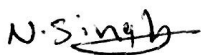
Jerin



Ian



Nirmal



Hamzah



Danyal



(c) *Software Development and Testing*

(i) **Software Development**

The software development stage can be broken down into many different stages which I will describe in chronological order. I will document the problems I encounter along the way and attempt to solve them, showing the solutions.

1. Creating the interface

To create the interface I first had to import Python's standard GUI toolkit, Tkinter. The whole code will be stored under one class: **adventureCircle** to represent the full game. I also set **RUN** to **false** so that the game would not play and just the layout would be processed first. From then I was able to make the frame in which more complicated widgets could be placed like labels and buttons on the game canvas. I set the colours and size of the canvas and added the play button and then added the title and instructions labels, and then the templates for the points and timer labels.

Python has three geometry managers: **pack**, **grid** and **place**. I first attempted to use the **place** manager to try and put each label exactly where I wanted it but it became too time consuming and the labels were not in line with each other in terms of the frame's dimensions.

For example, the timer and points label were both placed low down in the middle of the screen but were not aligned:

```
self.TimerText=Label(self.frame, bg="black", fg="white")
self.TimerText.place(x=150, y=260)
self.PointsText=Label(self.frame, bg="black", fg="white")
self.PointsText.place(x=150, y=270)
```

I decided to change geometry manager and went with the **pack** option which declares the position of widgets relative to each other. It is limited and the easiest to use of the three managers but is highly useful for my game. The updated labels:

```
self.TimerText=Label(self.frame, bg="black", fg="white")
self.TimerText.pack()
self.PointsText=Label(self.frame, bg="black", fg="white")
self.PointsText.pack()
```

To finish off I added a mainloop to make sure the program loops.

Code:

```
from tkinter import *
from random import *

class adventureCircle:

    def __init__(self):
        self.root=Tk()
        self.RUN=False

        self.frame=Frame(bg="black")
        self.frame.pack()

        self.Canvas=Canvas(self.frame, bg="black",width=300,height=300)
        self.Canvas.pack()

        self.TimerText=Label(self.frame, bg="black", fg="white")
        self.TimerText.pack()
        self.PointsText=Label(self.frame, bg="black", fg="white")
        self.PointsText.pack()
        self.Title=Label(self.frame, bg="black", fg="white", text="Welcome to adventureCircle!")
        self.Title.pack()
        self.Instructions=Label(self.frame, bg="black", fg="white", text="Click on the screen to move the
circle.")
        self.Instructions.pack()
        self.Instructions=Label(self.frame, bg="black", fg="white", text="Consume the green items and avoid
the red ones.")
        self.Instructions.pack()
        self.Instructions=Label(self.frame, bg="black", fg="white", text="Look out for yellow power-ups!")
        self.Instructions.pack()
        self.PlayButton=Button(self.frame, bg="black", fg="white", text="Play", command=self.start)
        self.PlayButton.pack()

        self.root.mainloop()
```

Here is how the interface looks:



2. Initialising the game and setting item positions

This subroutine is called start as it is invoked by the play button. I set **RUN** to **TRUE** because my code runs sequentially so that **RUN** is false when my interface is being created and **TRUE** during gameplay. I set the timer to equal zero so it starts counting from 0 as soon as the game is run and points to equal zero so that points starts from zero. I add the good, bad and power-up items with blank co-ordinates to allow the random integers to be output. The circle sprite will always start on the same point on the game canvas though: 100, 100 to lower the chance of an item being on the same position as the sprite. Next, I created identical co-ordinates to the circle sprite, **tempx** & **tempy** as this will be used to track collisions later on. I set all directions to **FALSE** so that the sprite doesn't move in any direction initially when play is clicked until the player moves it themselves. I set the initial size of the circle sprite to 3. Playing around with the sizes I felt that this would be the best size to use on the canvas. Finally, I set the canvas to recognise mouse clicks as the left mouse button click so that the circle spite can be moved. I put a command to initialise the **run** subroutine which will allow components to run like the timer and points I will create this actual subroutine later.

Code:

```
def start(self):
    self.Timer=0
    self.RUN=True

    self.GoodItemX=[]
    self.GoodItemY=[]

    self.BadItemX=[]
    self.BadItemY=[]

    self.PowerUpItemX=[[ ],[ ]]
    self.PowerUpItemY=[[ ],[ ]]

    self.Points=0

    self.x=100
    self.y=100
    self.tempx=100
    self.tempy=100
    self.UP=False
    self.DOWN=False
    self.LEFT=False
    self.RIGHT=False

    self.Size=3
    self.Canvas.bind("<ButtonPress-1>", self.MouseClicks)
    self.run()
```


3. On the game canvas

First of all I will create the actual circle sprite object. This is done easily as Tkinter has **create_shape** widgets so I used an oval for the sprite. I then set the conditions for the sprite to be active.

1 – If the timer is < 60

2 – If the size of the sprite is > 0

I then set the good and bad items to be created (subroutines for these will be created later). The code for the power-up item has to be different to the good and bad items or else it would pop up all the time. Finding the solution for this proved quite tricky. In the end I came up with this:

```
if randint(0,100)%2==0:
```

```
    self.CreatePowerUpItem(ball)
```

This code utilises the **randint** function and modulus divides the outputted integer by two. If the result is 0 then a power-up item can appear. I decided to do this because probability states that with this method a power-up item should appear half of the time but because it is completely random, some instances of gameplay the item will appear more and some it will appear less. I believe this surprise element will make the game more fun and unpredictable. I will now show a mini dry run just to demonstrate how it will work; I will do one test in which dry run will run until a power-up item is outputted.

% - modulus division (division where the remainder is the returned value)

!= - not equal to

Line	randint	%	Output	Comment
1	59			
2		2		
3			1	!= 0, no power-up item
4	80			
5		2		
6			0	== 0, power-up item displayed
7				

I then added the conditions for the game to end and they act as commands to initialise the end subroutine (created later).

- 1 – If the size of the sprite gets too big
- 2 – If the size of the sprite gets too small
- 3 – Else if the time reaches sixty seconds

These will also be tested later.

Code:

```
def paint(self):
    self.Canvas.delete(ALL)

    if self.Timer//100<=60:
        if 10*self.Size >0:
            ball=self.Canvas.create_oval(self.x-10*self.Size,self.y-
            10*self.Size,self.x+10*self.Size,self.y+10*self.Size, fill="white")
            self.CreateGoodItem(ball)
            self.CreateBadItem(ball)
            if randint(0,100)%2==0:
                self.CreatePowerUpItem(ball)
        elif 10*self.Size>600:
            self.TimerText['text']="You win! :D"
            self.end()
        else:
            self.TimerText['text']="You lose! :(
            self.end()
    else:
        self.TimerText['text']="Time's up!"
        self.end()
```

Here is what the circle sprite looks like on the game canvas so far:



4. Program runs and program ends

I started with the run subroutine. I set the condition that if **RUN = TRUE**:

- Timer starts counting in one's from zero
- Points is initialised and can increase or decrease depending on player actions
- Command to initialise the circle sprite movement subroutine (created later)
- Command to initialise the paint subroutine

Code:

```
def run(self):
    if self.RUN is True:
        self.Timer+=1
        self.TimerText['text']="Time: " + str(self.Timer//100)
        self.PointsText['text']="Points: " + str(self.Points)
        self.MoveCircleSprite(10*self.Size,2)
        self.paint()
        self.root.after(10, self.run)
```

The end subroutine is even simpler, all of the initialise commands are in other subroutines so this one literally just ends the program and the game. It sets **RUN** to **FALSE** and stops access to the left mouse button click on the game canvas.

Code:

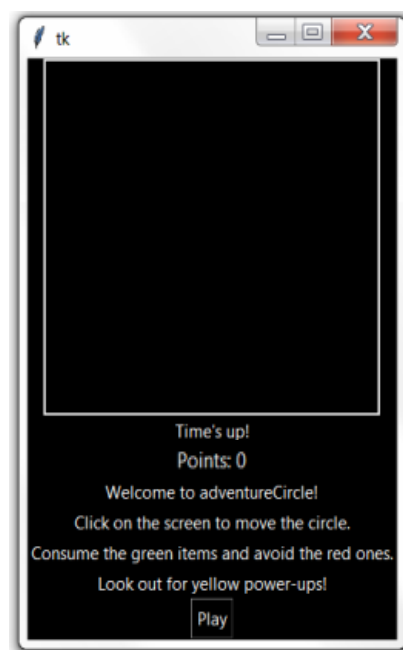
```
def end(self):
    self.RUN=False
    self.Canvas.unbind("<ButtonPress-1>")
```

In terms of alpha testing, I can now test that the timer and points label show and they are working. I can test that the starts counting immediately from when play is selected and the game ends on sixty seconds but I cannot test anything to do with points yet as the items are not coded yet.

This is proof that the timer is running as it should:



This is proof that the game ends when the timer has reached sixty seconds:



5. Circle sprite movement direction

Before I create the items, I want to make sure the circle sprite can actually move so that the **self.MouseClicks** command call actually works. I am going to code so that the sprite can move in eight directions (diagonal included) and hit off the walls.

Here is the original pseudocode algorithm:

```
tempx = eventx
tempy = eventy
IF eventx > selfx AND selfx IS NOT tempx THEN
    RIGHT = TRUE
    LEFT = FALSE
ELSEIF eventx < selfx AND selfx IS NOT tempx THEN
    LEFT = TRUE
    RIGHT = FALSE
ELSE THEN
    selfx = tempx
    RIGHT = FALSE
    LEFT = FALSE
IF eventy > selfy AND selfy IS NOT tempy THEN
    DOWN = TRUE
    UP = FALSE
ELSEIF eventy < selfy AND selfy IS NOT tempy THEN
    UP = TRUE
    DOWN = FALSE
ELSE THEN
    selfy = tempy
    DOWN = FALSE
    UP = FALSE
```

I will develop the algorithm in Python code. This is where the **temp** co-ordinates come into use. It is assigned to the **event** handler and that controls mouse clicks amongst many other things. (<**ButtonPress-1**> means left mouse button click.) I designed the code so that if **event** is more than **self** then the circle sprite moves to the right, e.g. if you click right on the game canvas away from the sprite. And vice versa with left. Same thing with the up and down direction. If they equal each other both directions are set to false.

Code:

```
def MouseClicks(self,event):
    self.tempx=event.x
    self.tempy=event.y
    if event.x> self.x and self.x is not self.tempx :
        self.RIGHT=True
        self.LEFT=False
    elif event.x< self.x and self.x is not self.tempx :
        self.LEFT=True
        self.RIGHT=False
    else:
        self.x=self.tempx
        self.RIGHT=False
        self.LEFT=False
    if event.y> self.y and self.y is not self.tempy :
        self.DOWN=True
        self.UP=False
    elif event.y< self.y and self.y is not self.tempy :
        self.UP=True
        self.DOWN=False
    else:
        self.y=self.tempy
        self.DOWN=False
        self.UP=False
```

Upon testing I have discovered that this code controls the direction of movement of the sprite, but not how it moves yet. I will develop this in the next subroutine.

6. Circle sprite movement and wall collisions

This was difficult to develop as I struggled to find a way to incorporate all aspects of the circle sprite movement together (e.g. the speed it goes, the direction it moves in and when it collides with each wall) but eventually I came up with a solution. I set **b** to be 10 * the size of the circle sprite, to represent the outer edges of the sprite. And so the actual co-ordinates of the sprite and **b** communicate with each other between the game canvas of [0, 300] to make sure the ball changes direction if it comes into contact with 0 or 300, the “walls”.

For example:

If the circle sprite was moving in and upward direction and its size was 4, **b** would be 40. The sprite would continue to move in its current direction (up). As soon as the co-ordinates hit 40, my code is set so that when the sprites co-ordinates minus **b** \leq zero then it changes direction, and that is what would happen.

I encountered a problem during development though. I didn't realise speed was treated as a vector quantity and had to have direction included:

```
if self.UP==True and self.y-b>0:
    self.y=speed
```

```
if self.DOWN==True and self.y+b<300:
    self.y=speed
```

The **y** minus **b** and **y** plus **b** have to match up to the speed so I added the direction just before the **=speed** code:

```
if self.UP==True and self.y-b>0:
    self.y-=speed
```

```
if self.DOWN==True and self.y+b<300:
    self.y+=speed
```

The **-=** assignment operator subtracts right operand from the left operand and assign the result to left operand and so **self.y-=speed** is equivalent to **self.y = self.y - speed**

The **+=** assignment operator adds right operand to the left operand and assign the result to left operand and so **self.y+=speed** is equivalent to **self.y = self.y + speed**

The circle sprite can now move in eight directions and handle wall collisions.

Code:

```
def MoveCircleSprite(self, b,speed):
    if self.UP==True and self.y-b>0:
        self.y-=speed
    elif self.UP==True and self.y-b<=0:
        self.UP=False
        self.DOWN=True
    if self.DOWN==True and self.y+b<300:
        self.y+=speed
    elif self.DOWN==True and self.y+b>=300:
        self.DOWN=False
        self.UP=True
    if self.LEFT==True and self.x-b>0:
        self.x-=speed
    elif self.LEFT==True and self.x-b<=0:
        self.LEFT=False
        self.RIGHT=True
    if self.RIGHT==True and self.x+b<300:
        self.x+=speed
    elif self.RIGHT==True and self.x+b>=300:
        self.RIGHT=False
        self.LEFT=True
```


7. Good item

This subroutine will contain the code for the good item and the instructions for how the ball will grow when the good item is consumed.

To make the items appearing more interesting, I developed code to add an extra item every fifteen seconds regardless of what is currently on the game canvas:

```
if len(self.GoodItemX) < self.Timer // 1500 + 1:
    self.GoodItemX.append(randint(50,250))
if len(self.GoodItemY) < self.Timer // 1500 + 1:
    self.GoodItemY.append(randint(50,250))
```

The timer is ticked by the millisecond and an item will appear between 50 and 250 of the game canvas. I started by setting it between 0 and 300 (the full game canvas) but this caused problems because as the circle sprite grew, it couldn't reach some items in the extreme co-ordinates so I made the range closer.

I started by using the **<find_enclosed>** collision handler as a method of consuming items but found upon alpha testing that this method only consumes the item when the item is inside the circle sprite. This could cause confusion among players as it is not what usually happens in these types of games and it would cause a delay in a points increase. It would cause a new item to not appear immediately also:

```
for i in range(0, len(self.GoodItemX)):
    if len(self.Canvas.<find_enclosed>(self.GoodItemX[i], self.GoodItemY[i], self.GoodItemX[i]+10,
self.GoodItemY[i]+10)) is not 1:
        if ball in self.Canvas.<find_enclosed>(self.GoodItemX[i], self.GoodItemY[i], self.GoodItemX[i]+10,
self.GoodItemY[i]+10):
```

To solve this problem, I changed **<find_enclosed>** to **find_overlapping** so that the item will be consumed as soon as it touches the circle sprite and a new item will appear in a random position on the screen immediately:

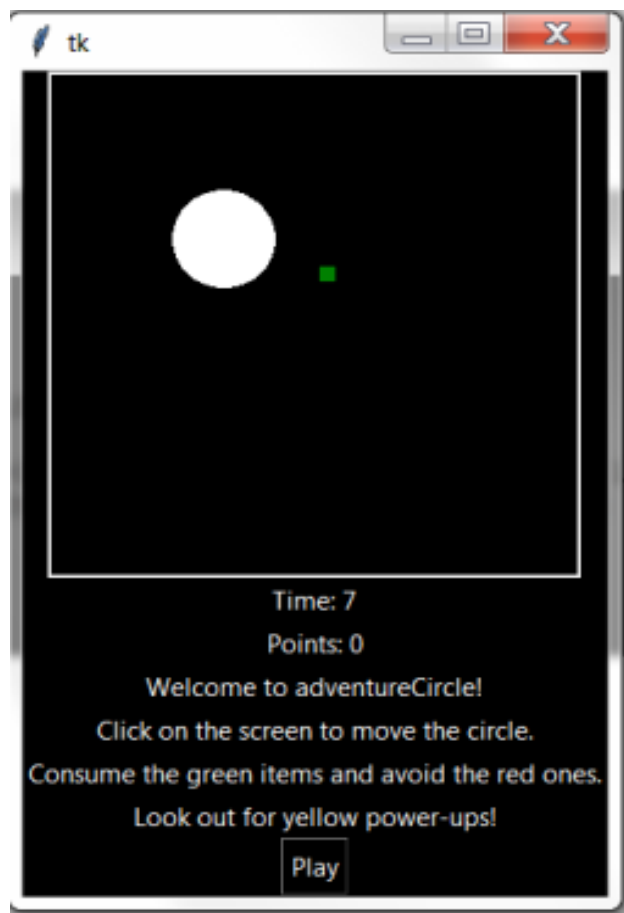
```
for i in range(0, len(self.GoodItemX)):
    if len(self.Canvas.find_overlapping(self.GoodItemX[i], self.GoodItemY[i], self.GoodItemX[i]+10,
self.GoodItemY[i]+10)) is not 1:
        if ball in self.Canvas.find_overlapping(self.GoodItemX[i], self.GoodItemY[i], self.GoodItemX[i]+10,
self.GoodItemY[i]+10):
```

I then added code so that each good item consumed will cause an increase in points of 100 and an increase in the circle sprite size by 0.5.

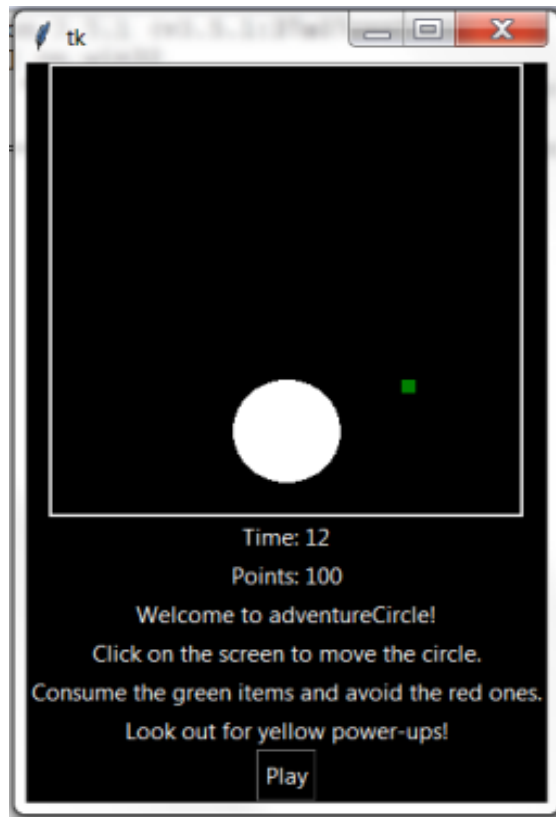
Code:

```
def CreateGoodItem(self,ball):
    if len(self.GoodItemX) < self.Timer//1500 +1:
        self.GoodItemX.append(randint(50,250))
    if len(self.GoodItemY) < self.Timer//1500 +1:
        self.GoodItemY.append(randint(50,250))
    for i in range(0,len(self.GoodItemX)):
        self.Canvas.create_rectangle(self.GoodItemX[i], self.GoodItemY[i], self.GoodItemX[i]+10,
self.GoodItemY[i]+10, fill="green")
    for i in range(0,len(self.GoodItemX)):
        if len(self.Canvas.find_overlapping(self.GoodItemX[i], self.GoodItemY[i], self.GoodItemX[i]+10,
self.GoodItemY[i]+10)) is not 1:
            if ball in self.Canvas.find_overlapping(self.GoodItemX[i], self.GoodItemY[i], self.GoodItemX[i]+10,
self.GoodItemY[i]+10):
                self.Points+=100
                self.Size+=0.5
                self.GoodItemX.pop(i)
                self.GoodItemY.pop(i)
                self.CreateGoodItem(ball)
```

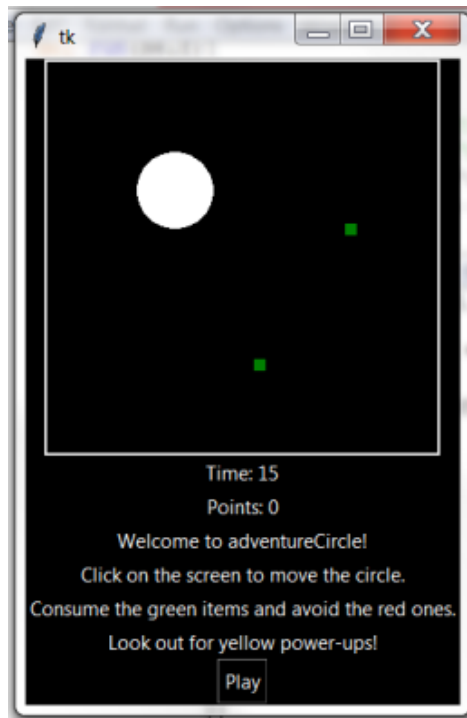
Here is how the game looks with the code for good items implemented:



Adds 100 points and increases the circle sprite size by 0.5 when consumed:



New item after 15 seconds:



8. Bad item

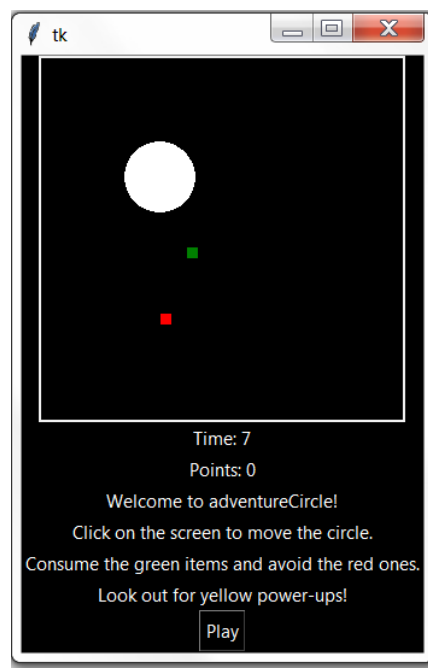
The code for the bad item is almost identical to the code for the good item apart from 3 things.

- 1 – The colour is red
- 2 – Points will decrease by 50
- 3 – Size will decrease by 0.5

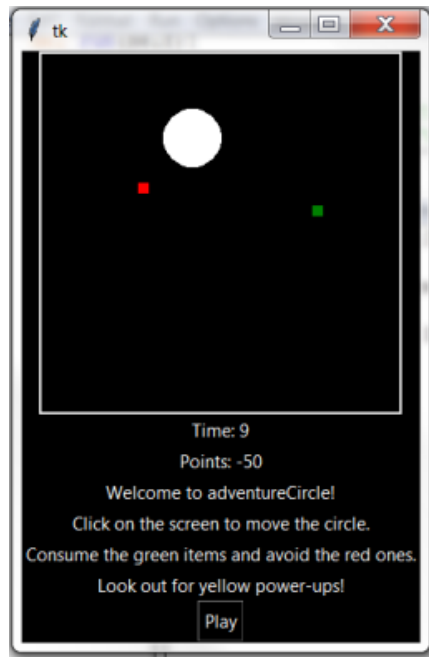
Code:

```
def CreateBadItem(self,ball):
    if len(self.BadItemX) < self.Timer//1500 +1:
        self.BadItemX.append(randint(50,250))
    if len(self.BadItemY) < self.Timer//1500 +1:
        self.BadItemY.append(randint(50,250))
    for i in range(0,len(self.BadItemX)):
        self.Canvas.create_rectangle(self.BadItemX[i],          self.BadItemY[i],          self.BadItemX[i]+10,
self.BadItemY[i]+10, fill="red")
    for i in range(0,len(self.BadItemX)):
        if len(self.Canvas.find_overlapping(self.BadItemX[i],    self.BadItemY[i],    self.BadItemX[i]+10,
self.BadItemY[i]+10)) is not 1:
            if ball in self.Canvas.find_overlapping(self.BadItemX[i], self.BadItemY[i], self.BadItemX[i]+10,
self.BadItemY[i]+10):
                self.Points-=50
                self.Size-=0.5
                self.BadItemX.pop(i)
                self.BadItemY.pop(i)
                self.CreateBadItem(ball)
```

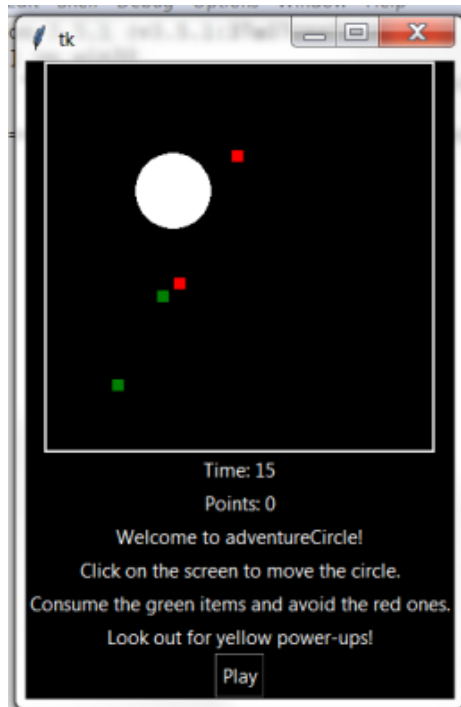
Here is how the game looks with the code for bad items implemented:



Takes away 50 points and decreases the circle sprite size by 0.5 when consumed:



New item after 15 seconds:



9. Power-up item

I coded the power-up item to appear every ten seconds, and that is the only condition. To signify it is a power-up I made it flash by using the for loop and setting it to zero and then initialising it so it becomes a recursive function and keeps running hence the flashing.

```
self.PowerUpItemX[0].append(randint(50,250))
self.PowerUpItemY[0].append(randint(50,250))
```

```
self.PowerUpItemX[0].pop(i)
self.PowerUpItemY[0].pop(i)
```

Apart from that the code is the same as the good and bad item codes apart from:

- 1 – The colour is yellow
- 2 – Points will increase by 150
- 3 – Size will increase by 1

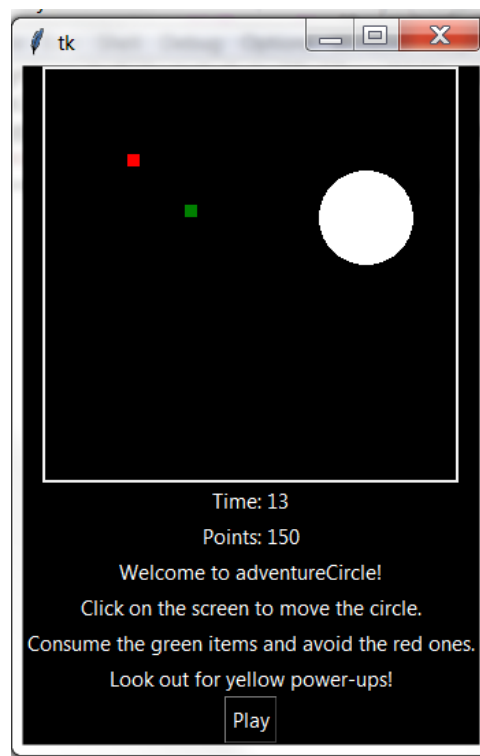
Code:

```
def CreatePowerUpItem(self,ball):
    if len(self.BadItemY) is 0 or self.Timer%1000 == 0 :
        self.PowerUpItemX[0].append(randint(50,250))
        self.PowerUpItemY[0].append(randint(50,250))
    for i in range(0,len(self.PowerUpItemX[0])):
        self.Canvas.create_rectangle(self.PowerUpItemX[0][i],
                                     self.PowerUpItemY[0][i],
                                     self.PowerUpItemX[0][i]+10, self.PowerUpItemY[0][i]+10, fill="yellow")
    for i in range(0,len(self.PowerUpItemX[0])):
        if len(self.Canvas.find_overlapping(self.PowerUpItemX[0][i],
                                             self.PowerUpItemX[0][i]+10, self.PowerUpItemY[0][i]+10)) is not 1:
            if ball in self.Canvas.find_overlapping(self.PowerUpItemX[0][i],
                                                     self.PowerUpItemX[0][i]+10, self.PowerUpItemY[0][i]+10):
                self.Points+=150
                self.Size+=1
                self.PowerUpItemX[0].pop(i)
                self.PowerUpItemY[0].pop(i)
                self.CreatePowerUpItem(ball)
```

Here is how the game looks with the code for power-up items implemented:



Adds 150 points and increases the circle sprite size by 1 when consumed:



The full and annotated code for the game is shown here.

Code of game

```
#This is the full annotated code for adventureCirle

#These import function call Tkinter, Python's standard GUI
#and random, to generate random numbers
from tkinter import *
from random import *

#The game is held under one class
class adventureCircle:

    #The subroutine that creates the interface and places the widgets
    def __init__(self):
        self.root=Tk()
        self.RUN=False

        self.frame=Frame(bg="black")
        self.frame.pack()

        self.Canvas=Canvas(self.frame, bg="black",width=300,height=300)
        self.Canvas.pack()

        self.root.resizable(0, 0)

        self.TimerText=Label(self.frame, bg="black", fg="white")
        self.TimerText.pack()
        self.PointsText=Label(self.frame, bg="black", fg="white")
        self.PointsText.pack()
        self.Title=Label(self.frame, bg="black", fg="white", text="Welcome to
adventureCircle!")
        self.Title.pack()
        self.Instructions=Label(self.frame, bg="black", fg="white", text="Click on the screen
to move the circle.")
        self.Instructions.pack()
        self.Instructions=Label(self.frame, bg="black", fg="white", text="Consume the green
items and avoid the red ones.")
        self.Instructions.pack()
        self.Instructions=Label(self.frame, bg="black", fg="white", text="Look out for yellow
power-ups!")
        self.Instructions.pack()
        self.PlayButton=Button(self.frame, bg="black", fg="white", text="Play"
,command=self.start)
        self.PlayButton.pack()

        self.root.mainloop()

    #The subroutine that initialises all components when the play button is pressed
    def start(self):
        self.Timer=0
        self.RUN=True

        self.GoodItemX=[]
        self.GoodItemY=[]

        self.BadItemX=[]
        self.BadItemY=[]

        self.PowerUpItemX=[[[]],[[]]]
        self.PowerUpItemY=[[[]],[[]]]

        self.Points=0

        self.x=100
        self.y=100
        self.tempx=100
        self.tempy=100
        self.UP=False
        self.DOWN=False
```



```

self.LEFT=False
self.RIGHT=False

self.Size=3
self.Canvas.bind("<ButtonPress-1>", self.MouseClicks)
self.run()

def run(self):
    if self.RUN is True:
        self.Timer+=1
        self.TimerText['text']="Time: " + str(self.Timer//100)
        self.PointsText['text']="Points: " + str(self.Points)
        self.MoveCircleSprite(10*self.Size,2)
        self.paint()
        self.root.after(10, self.run)

#Subroutine that ends the game once it is game over
def end(self):
    self.RUN=False
    self.Canvas.unbind("<ButtonPress-1>")

#Subroutine for the good item
def CreateGoodItem(self,ball):
    if len(self.GoodItemX) <self.Timer//1500 +1:
        self.GoodItemX.append(randint(50,250))
    if len(self.GoodItemY) <self.Timer//1500 +1:
        self.GoodItemY.append(randint(50,250))
    for i in range(0,len(self.GoodItemX)):
        self.Canvas.create_rectangle(self.GoodItemX[i], self.GoodItemY[i],
self.GoodItemX[i]+10, self.GoodItemY[i]+10, fill="green")
    for i in range(0,len(self.GoodItemX)):
        if len(self.Canvas.find_overlapping(self.GoodItemX[i], self.GoodItemY[i],
self.GoodItemX[i]+10, self.GoodItemY[i]+10)) is not 1:
            if ball in self.Canvas.find_overlapping(self.GoodItemX[i], self.GoodItemY[i],
self.GoodItemX[i]+10, self.GoodItemY[i]+10):
                self.Points+=100
                self.Size+=0.5
                self.GoodItemX.pop(i)
                self.GoodItemY.pop(i)
                self.CreateGoodItem(ball)

#Subroutine for the bad item
def CreateBadItem(self,ball):
    if len(self.BadItemX) <self.Timer//1500 +1:
        self.BadItemX.append(randint(50,250))
    if len(self.BadItemY) <self.Timer//1500 +1:
        self.BadItemY.append(randint(50,250))
    for i in range(0,len(self.BadItemX)):
        self.Canvas.create_rectangle(self.BadItemX[i], self.BadItemY[i],
self.BadItemX[i]+10, self.BadItemY[i]+10, fill="red")
    for i in range(0,len(self.BadItemX)):
        if len(self.Canvas.find_overlapping(self.BadItemX[i], self.BadItemY[i],
self.BadItemX[i]+10, self.BadItemY[i]+10)) is not 1:
            if ball in self.Canvas.find_overlapping(self.BadItemX[i], self.BadItemY[i],
self.BadItemX[i]+10, self.BadItemY[i]+10):
                self.Points-=50
                self.Size-=0.5
                self.BadItemX.pop(i)
                self.BadItemY.pop(i)
                self.CreateBadItem(ball)

#Subroutine for the power-up item
def CreatePowerUpItem(self,ball):
    if len(self.BadItemY) is 0 or self.Timer%1000 == 0 :
        self.PowerUpItemX[0].append(randint(50,250))
        self.PowerUpItemY[0].append(randint(50,250))
    for i in range(0,len(self.PowerUpItemX[0])):
        self.Canvas.create_rectangle(self.PowerUpItemX[0][i], self.PowerUpItemY[0][i],
self.PowerUpItemX[0][i]+10, self.PowerUpItemY[0][i]+10, fill="yellow")
    for i in range(0,len(self.PowerUpItemX[0])):
        if len(self.Canvas.find_overlapping(self.PowerUpItemX[0][i],
self.PowerUpItemY[0][i], self.PowerUpItemX[0][i]+10, self.PowerUpItemY[0][i]+10)) is not 1:
            if ball in self.Canvas.find_overlapping(self.PowerUpItemX[0][i],
self.PowerUpItemY[0][i], self.PowerUpItemX[0][i]+10, self.PowerUpItemY[0][i]+10):

```

```

        self.Points+=150
        self.Size+=1
        self.PowerUpItemX[0].pop(i)
        self.PowerUpItemY[0].pop(i)
        self.CreatePowerUpItem(ball)

#Subroutine for the circle sprite and the game over conditions
def paint(self):
    self.Canvas.delete(ALL)

    if self.Timer//100<=60:
        if 10*self.Size >0:
            ball=self.Canvas.create_oval(self.x-10*self.Size,self.y-
10*self.Size,self.x+10*self.Size,self.y+10*self.Size, fill="white")
            self.CreateGoodItem(ball)
            self.CreateBadItem(ball)
            if randint(0,100)%2==0:
                self.CreatePowerUpItem(ball)
        elif self.Size>=6:
            self.TimerText['text']="You win! :D"
            self.end()
        else:
            self.TimerText['text']="You lose! :("
            self.end()
    else:
        self.TimerText['text']="Time's up!"
        self.end()

#Subroutine for the wall collisions of the circle sprite
def MoveCircleSprite(self, b,speed):
    if self.UP==True and self.y-b>0:
        self.y-=speed
    elif self.UP==True and self.y-b<=0:
        self.UP=False
        self.DOWN=True
    if self.DOWN==True and self.y+b<300:
        self.y+=speed
    elif self.DOWN==True and self.y+b>=300:
        self.DOWN=False
        self.UP=True
    if self.LEFT==True and self.x-b>0:
        self.x-=speed
    elif self.LEFT==True and self.x-b<=0:
        self.LEFT=False
        self.RIGHT=True
    if self.RIGHT==True and self.x+b<300:
        self.x+=speed
    elif self.RIGHT==True and self.x+b>=300:
        self.RIGHT=False
        self.LEFT=True

#Subroutine to control the direction of movement of the circle sprite
#(through mouse clicks on the game canvas)
def MouseClicks(self,event):
    self.tempx=event.x
    self.tempy=event.y
    if event.x> self.x and self.x is not self.tempx :
        self.RIGHT=True
        self.LEFT=False
    elif event.x< self.x and self.x is not self.tempx :
        self.LEFT=True
        self.RIGHT=False
    else:
        self.x=self.tempx
        self.RIGHT=False
        self.LEFT=False
    if event.y> self.y and self.y is not self.tempy :
        self.DOWN=True
        self.UP=False
    elif event.y< self.y and self.y is not self.tempy :
        self.UP=True
        self.DOWN=False
    else:
        self.y=self.tempy

```

```
self.DOWN=False  
self.UP=False
```

```
app=adventureCircle()
```

(ii) Testing

I am now going to test the game. As mentioned in my test strategy, these are the types of testing I intended to do:



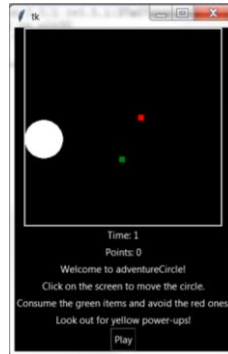

❖ Alpha testing

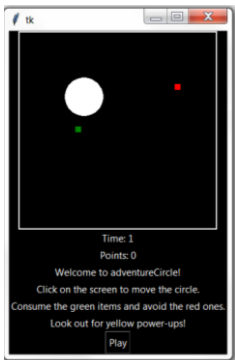
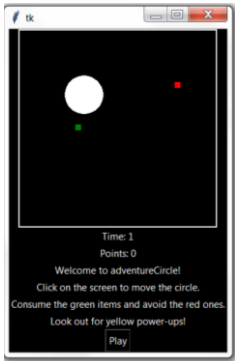
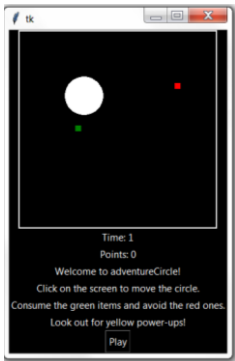

- The functionality of the game is tested by me on code that is just written. This will help me find any bugs in the code so far and fix them.
- In alpha testing, I will perform three types of tests in the system:
 - **Black box** – This involves checking the output of the game and see if it matches with the expected output in the test strategy. In each test, there is normal data (data that normally works), extreme data (data that is on the borderline of working) and abnormal data (data that is normally rejected) so the game will function correctly.
 - **Functionality** – This tests that the game runs correctly and modules link together.
 - **Navigation** – This checks that the navigation links are working and screens will be switched correctly.


❖ Beta testing

- This is performed by my end users. The code used will be a near finished product that is almost ready to be released. My end users could possibly find bugs within the game, and provide comments on the game and make any suggestions to improving the game.

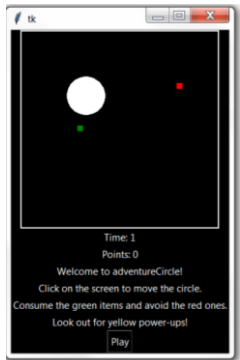

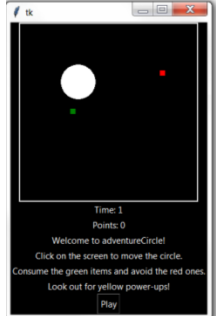
Alpha Testing


Test no.	Test description	Outcome
Tests involving movement of the circle sprite		
1	Up movement of circle sprite using valid left mouse button click above circle sprite	<p>Success</p> 
2	Down movement of circle sprite using valid left mouse button click below circle sprite	<p>Success</p> 
3	Left movement of circle sprite using valid left mouse button click to the left of circle sprite	<p>Success</p> 
4	Right movement of circle sprite using valid left mouse button click to the right of circle sprite	<p>Success</p> 

5	No movement of circle sprite with no player input	<p>Success</p> 
6	No movement of circle sprite when edges of circle sprite clicked	<p>Success</p> 
7	No movement of circle sprite when edges of game canvas (walls) are clicked	<p>Success</p> 
8	No movement of circle sprite when right mouse button is clicked	<p>Success</p> 

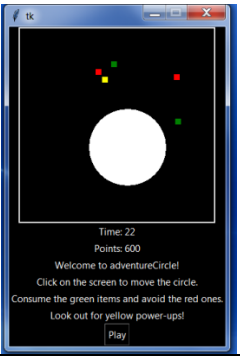

9	No movement of circle sprite when letter key is pressed (W)	<p>Success</p> 
---	---	---

Tests involving game navigation

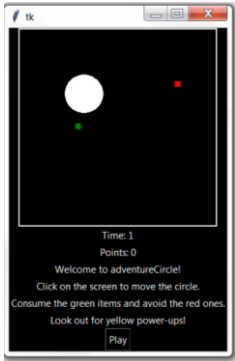



10	Display interface on game canvas with valid left mouse button click on play button	<p>Success – game appeared</p> 
11	Display interface on game canvas with long holding (ten seconds) left mouse button click on play button	<p>Success – game appeared</p> 
12	Display interface on game canvas with valid left mouse button click on edge of play button	<p>Success – game appeared</p> 

13	Display interface on game canvas with invalid right mouse button click on play button	<p>Success – game did not appear</p> 
----	---	---

Tests involving points system

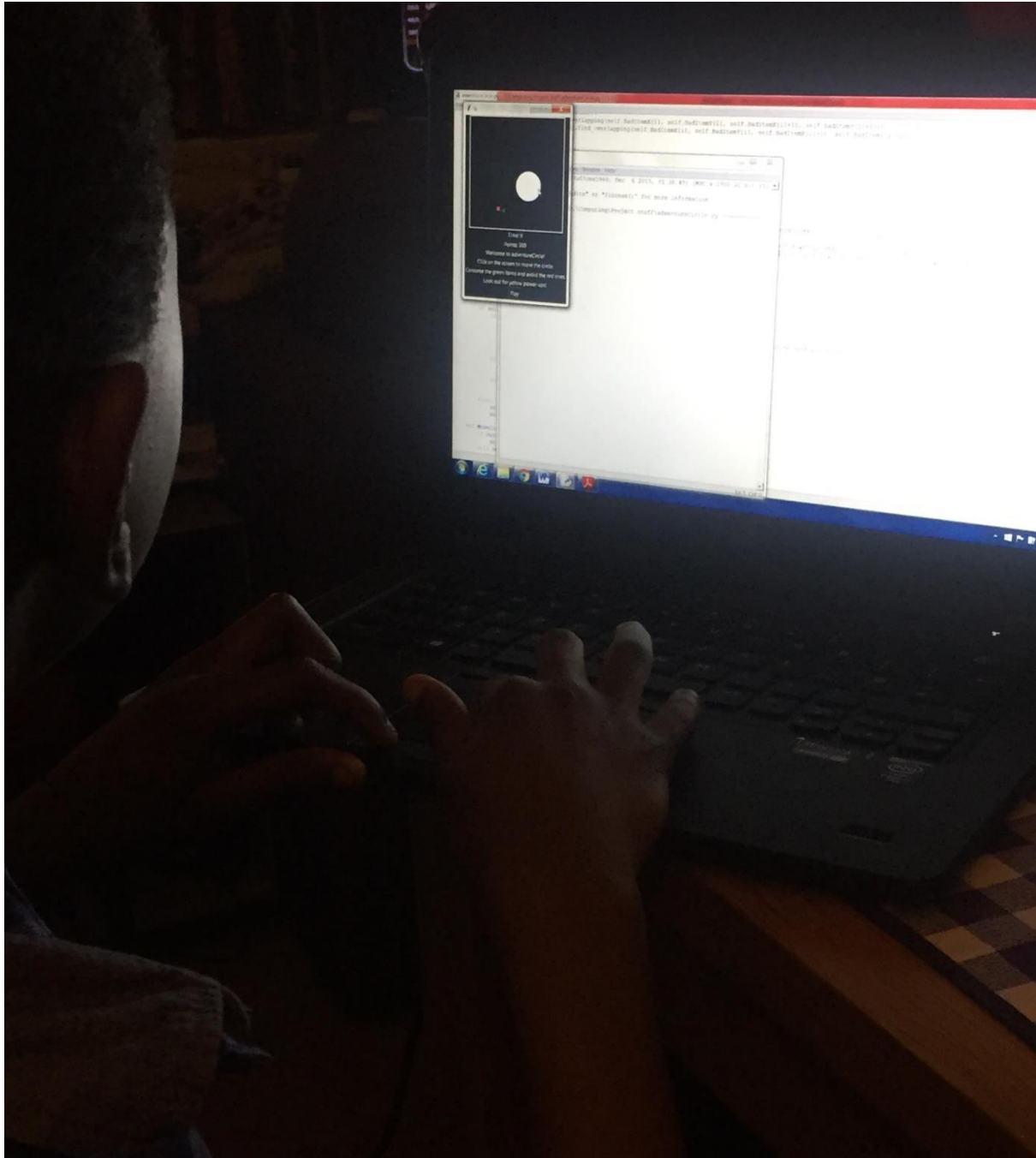
14	Consume six consecutive good items to acquire 600 points and end the game with a “You win” message	<p>Failed – the game didn’t end</p> 
15	Consume six consecutive bad items to acquire -300 points and end the game with a “You lose” message	<p>Success – game ended and message displayed</p> 

Tests involving timer system

16	Immediate starting of the timer	<p>Success</p> 
17	Timer should count in seconds one second after the other	<p>Success</p> 
18	Timer should stop immediately 60 seconds is reached	<p>Success</p> 
19	Change timer functionality by clicking on it with the left mouse button	<p>Success – Nothing happened</p> 

Beta testing

The first beta test was carried out by my brother and his four friends who made up the primary end users. His friends came round to our house and they all had turns playing the game. Here is evidence of primary end user beta testing:



I asked them to tell me the main positives and negatives of the game after gameplay.

Positives

- The game is fun and exciting to play
- The power-up feature is good
- Easy to navigate
- Good, simple colour scheme
- Not too easy, not too hard

Negatives

- Easy to click off the screen and navigate away from the game
- Changing the size makes it look aesthetically bad (e.g. full screen)

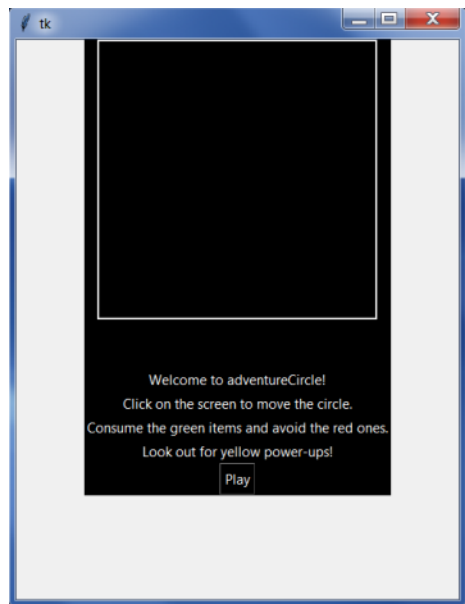
After testing, I see that my game has been a major success as all of the primary end users like it. However, they have picked up on some minor negatives which I will attempt to address.

Easy to click off the screen and navigate away from the game

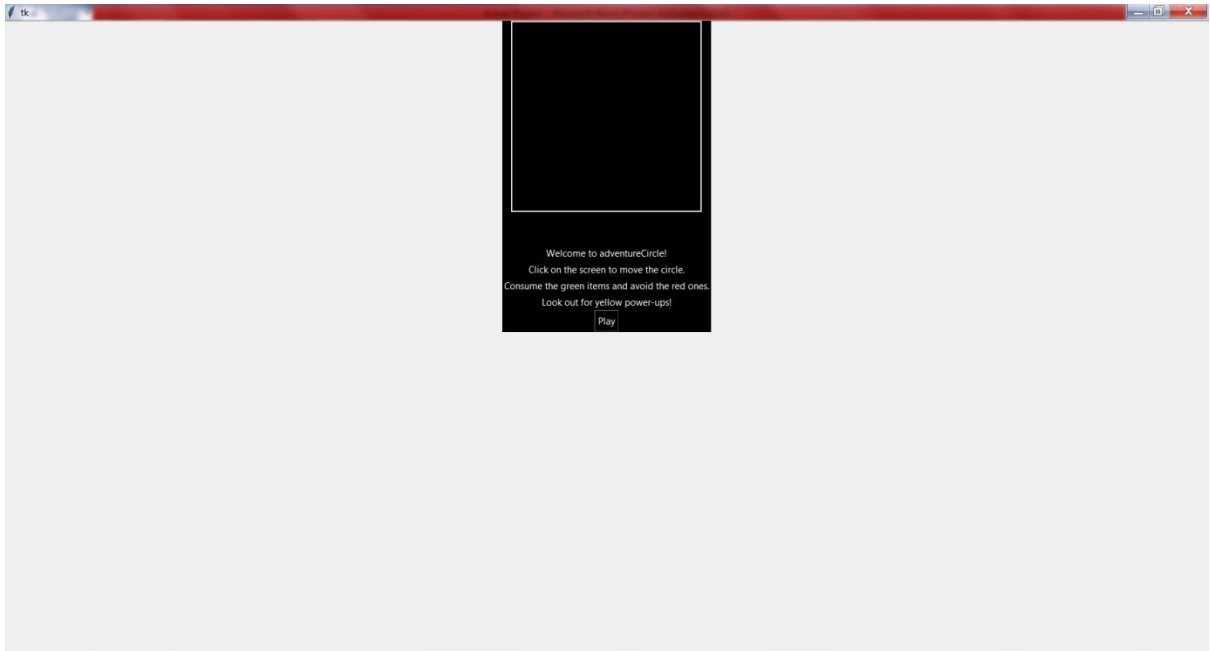
This is a problem which I cannot solve internally as it is not a problem within the coding. All I can suggest is to move the game to somewhere that will be easier to click on the screen. Or increase the size of the screen to make it so that if the player clicks off the game screen the interface won't close. This brings me on to the next problem.

Changing the size makes it look aesthetically bad (e.g. full screen)

I realised, while observing the beta testing that the frame/canvas is fixed, and so increasing the screen size in any way will not alter the interface:



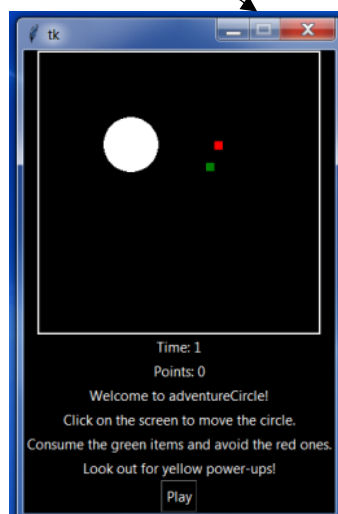
Full screen:



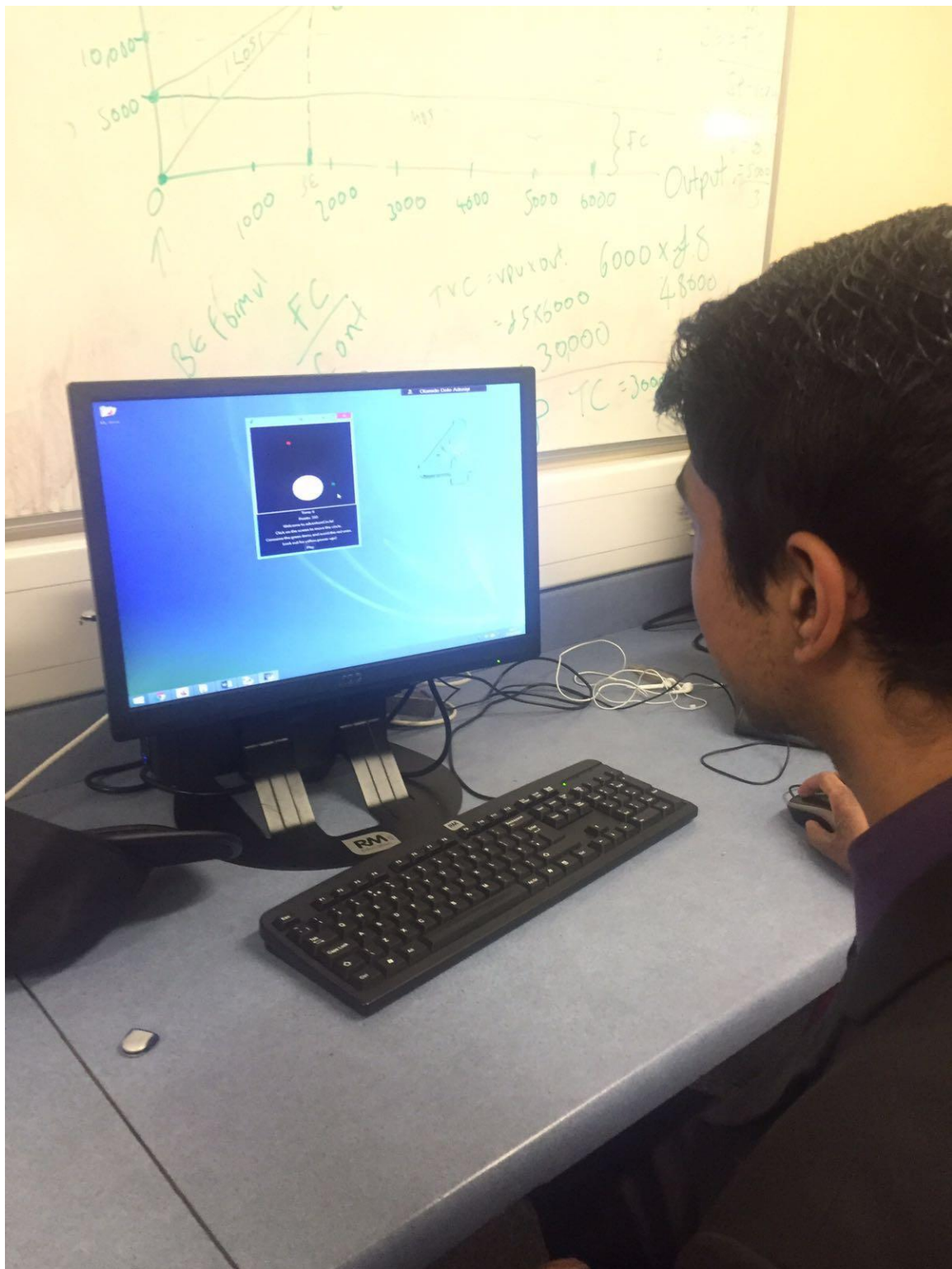
To solve this issue, I will add code to `__init__(self)` subroutine so that the canvas cannot be changed.

The code that is added is `self.root.resizable(0, 0)`

This has now resolved the issue.



The second beta test was carried out by my other end users (my classmates). This was done in class. Here is evidence of end user beta testing:



I asked them to tell me the main positives and negatives of the game after gameplay.

Positives

- The game is fun and exciting to play
- The circle sprite moves accurately with the mouse's click
- Good colours used for the items

Negatives

- Background colour too dark
- Game crashes at the end

Again after testing, I can conclude that my game has been a major success as **ALL** of the end users like it. However, again they have picked up on some negatives which I will address.

Background colour too dark

This was only mentioned by one user and I can treat it as an anomaly because in the second questionnaire that I conducted. The majority vote was for a classic colour scheme and all of my end users signed to the agreement of my requirements and design specification which included the mention of a black background.

Game crashes at the end

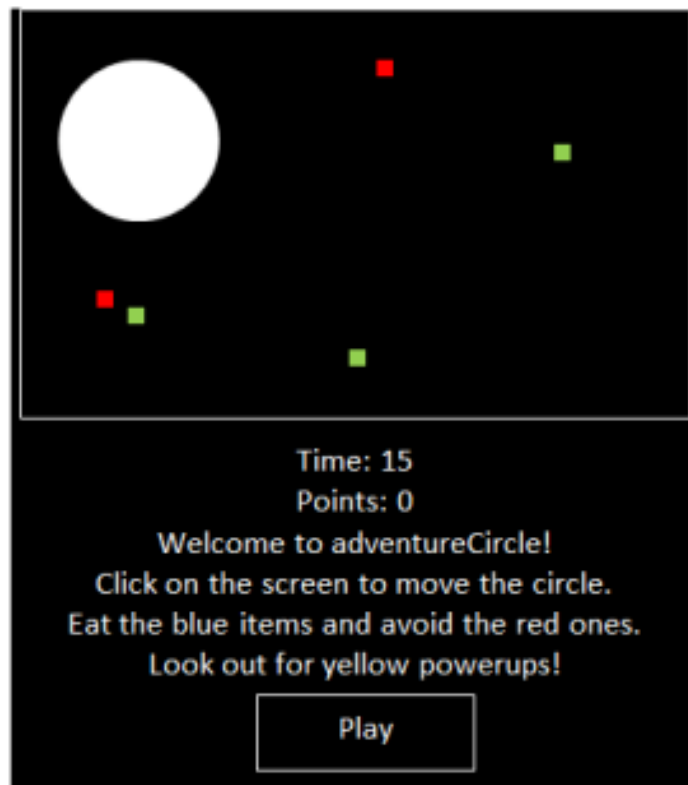
This was picked up on by myself also in my alpha testing (test 14) which involves the points system and the points needed to win the game. I will address this at a later time.

(d) Documentation

I am going to create a user guide for my game so players can know where to get the game from, find solutions to any issues they might have and see what requirements they would need to run the game.



adventureCircle



User Guide

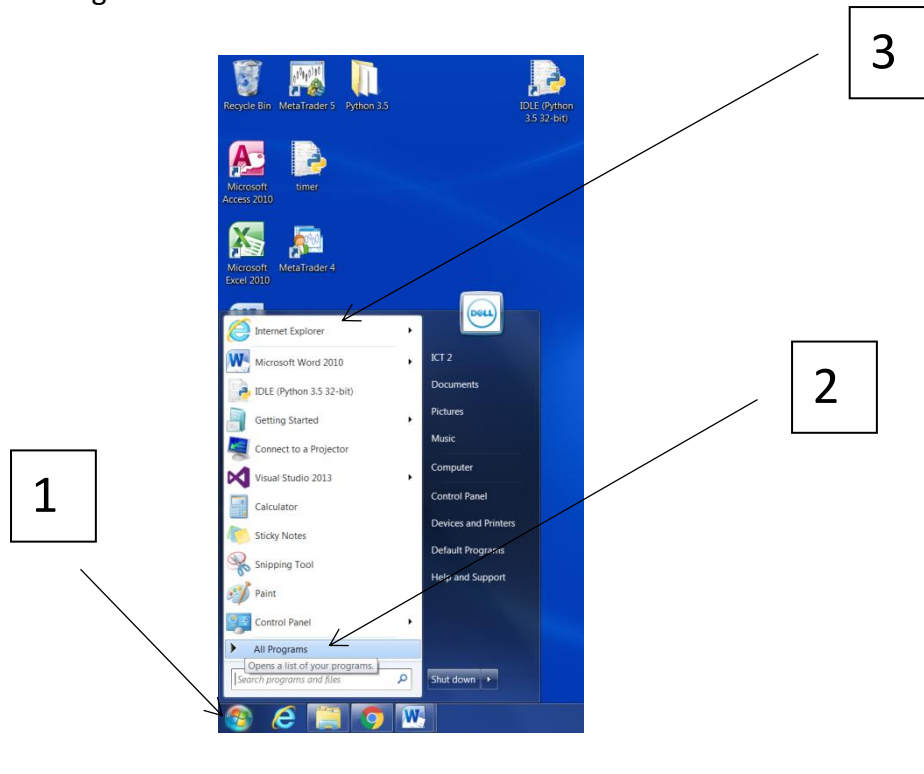
Contents

Download.....	2
Starting The Game.....	4
Navigation.....	5
How To Play.....	6
Troubleshooting.....	8
Backup.....	9

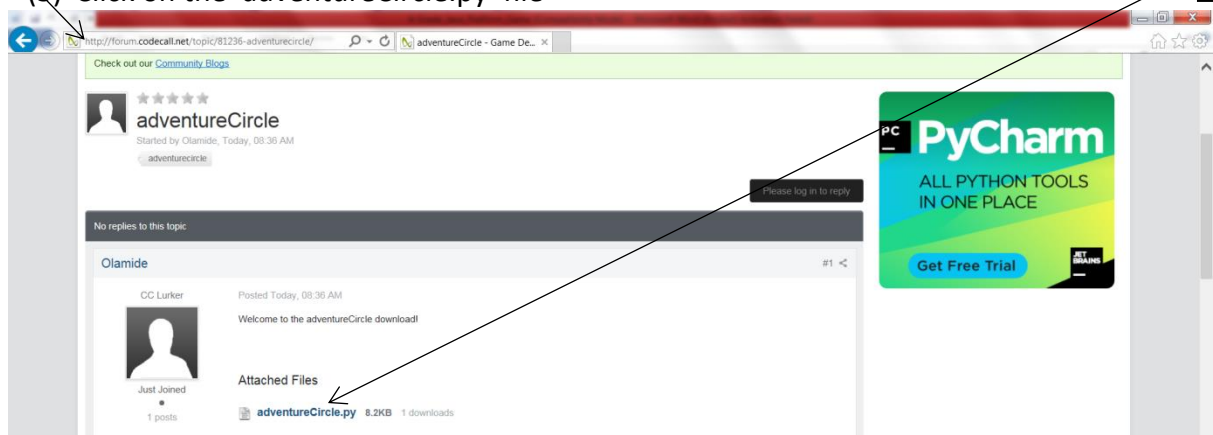
Download

To download the game, follow these steps:

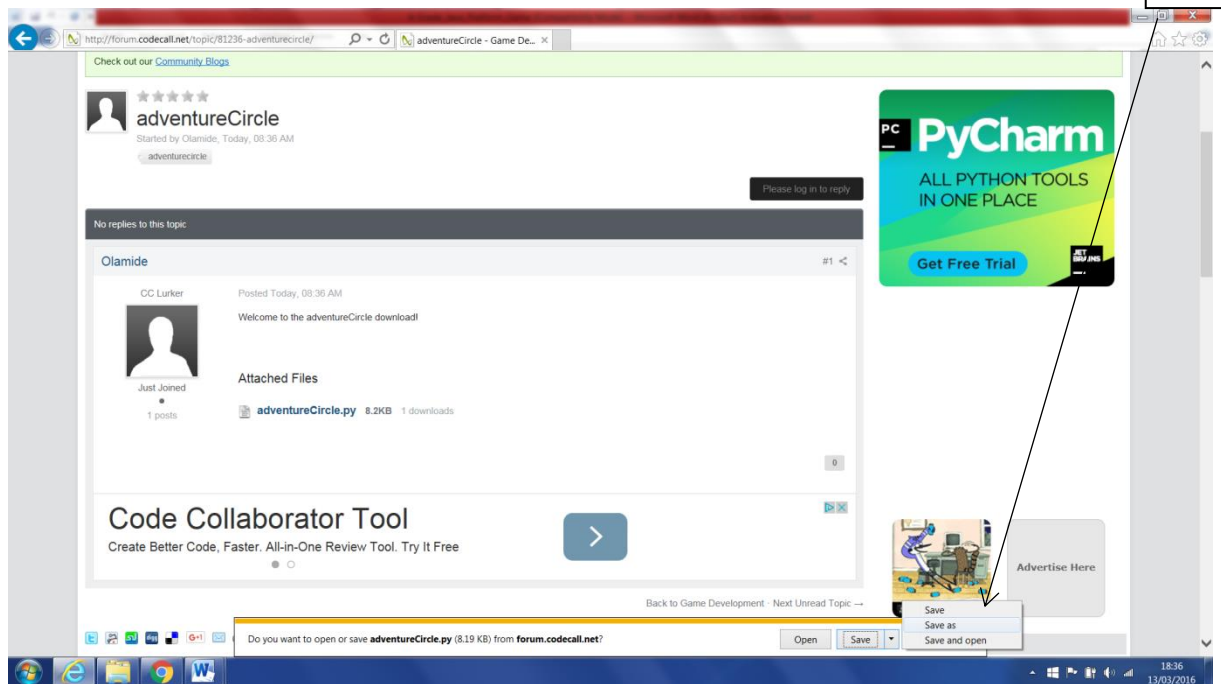
- (1) Click 'Start' which is at the bottom left of your screen
- (2) Click 'All Programs'
- (3) Open Internet Explorer / Google Chrome / Mozilla Firefox / Apple Safari or any other working browser



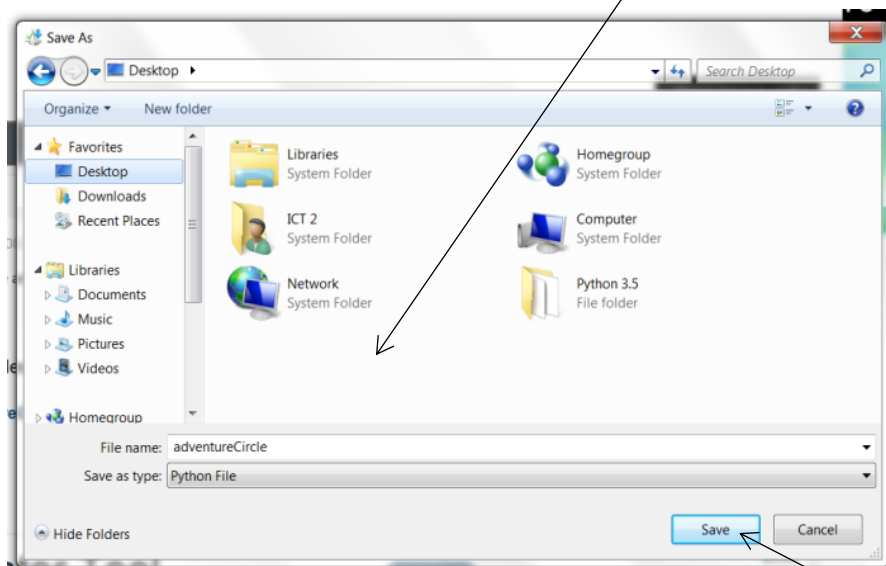
- (4) Type <http://forum.codecall.net/topic/81236-adventurecircle/> into the browser's address bar and press enter
- (5) Click on the 'adventureCircle.py' file



- (6) Click the arrow next to the 'Save' button and then select the 'Save as' option at the bottom of the screen



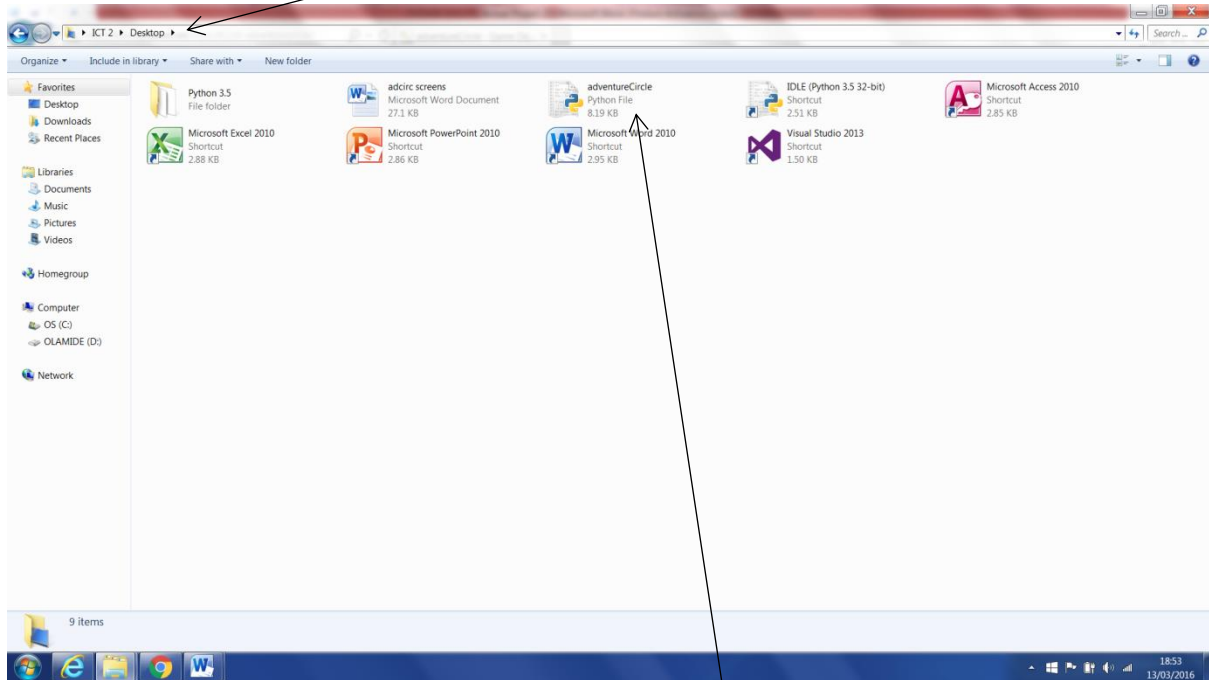
- (7) Select where to save the game
(8) Click save



Starting The Game

- (1) Find where you saved the game
- (2) Double click on the icon

1

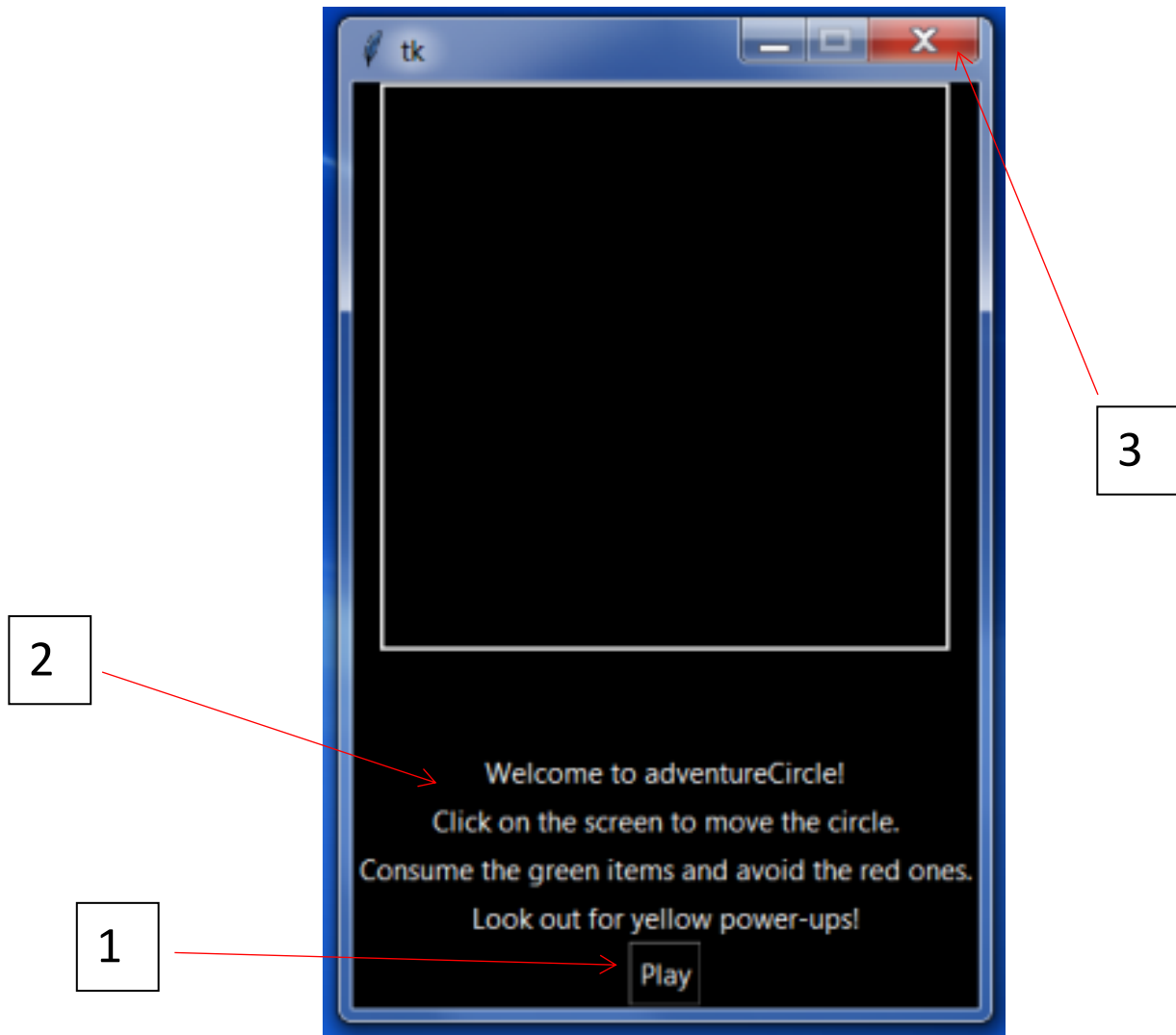


2

Navigation

Main Menu Screen

- (1) This is the play button. When clicked, the game will initialise and start a new game.
- (2) These are the instructions, they tell you how to play the game
- (3) Click this to exit the game once done playing

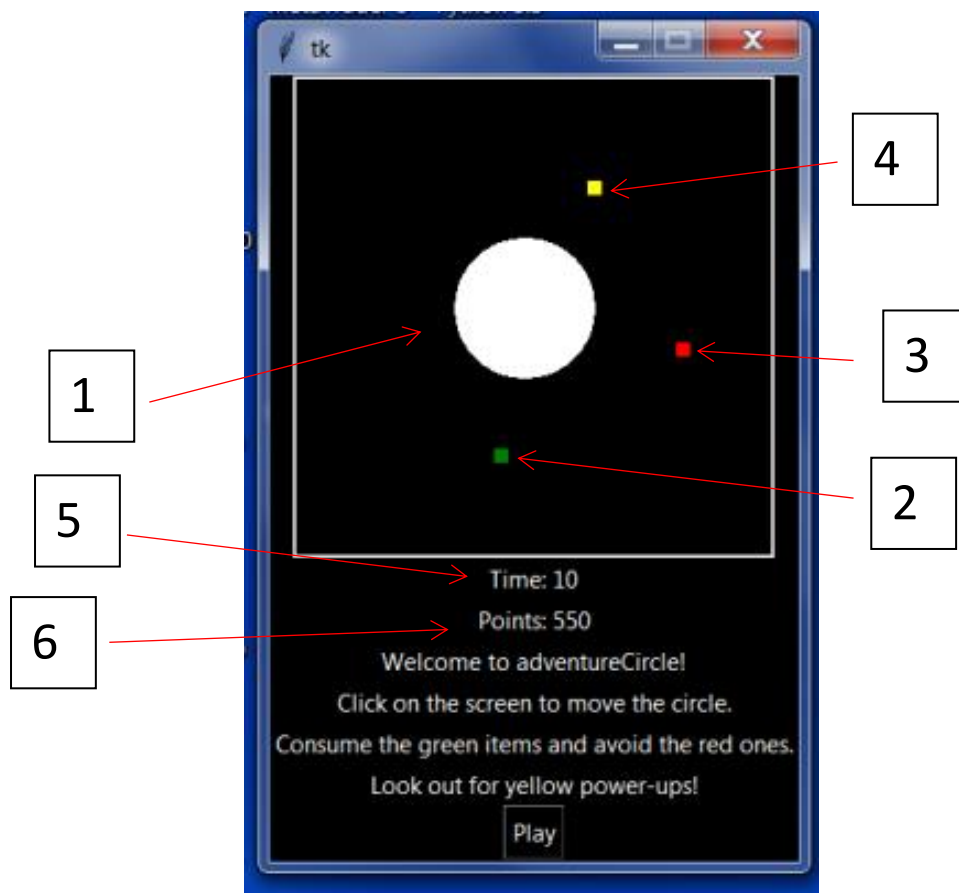


How To Play

To control the movement of the circle sprite, click anywhere on the game screen. If you click above the circle sprite it will move upward and vice versa with below the sprite. If you click to the left of the sprite it will move leftward and vice versa with clicking to the right.

The game screen is displayed below.

- (1) This is the circle sprite
- (2) This is a good item. If consumed it adds 100 to your current points and increases the size of the circle sprite
- (3) This is a bad item. If consumed it takes away 50 from your current points and decreases the size of the circle sprite
- (4) This is a power-up item. If consumed it adds 150 to your current points and increases the size of the circle sprite by more than the good item
- (5) This is the timer. It counts from zero and ends the game if it reaches sixty
- (6) This is the points label. It displays current points



The objective of the game is to have as many points as possible before the game ends by consuming as many good and power-up items as possible and avoiding the bad items.

The game is won when the circle sprite gets too big.

The game is lost when the circle sprite gets too small.

Troubleshooting

If you have any issues around the game, look to and see if any of these apply to you. If so, there are possible causes and solutions to help you.

Download Issues

Issue	Possible Cause	Solution
Unable to access the website	You are not connected to the internet	Contact your internet service provider
	The website is down for maintenance	Try again at a later time
Game cannot be downloaded	Insufficient disk space in hard drive	Make you have enough space in your hard drive to store the game, 25MB in the minimum requirement

General Issues

Issue	Possible Cause	Solution
Game does not open	Your current operating system may not be appropriate of running the game	Check that your operating system is of Microsoft Windows Vista or any later operating systems in used
	You may not have Python or the correct Python version needed to run the game	Download the latest version at https://www.python.org/downloads/
Game is running too slow and it lags	You do not have enough RAM	Make sure you run this game on a computer that has at least 512 MB of RAM which is what is recommended
	You do not have enough processing power	Make sure the computer that runs this game has a total processor with a clock rate of 1.2GHz or more
	Game is out of your screen	Drag your game using the bar at the top of the window so all of it is visible

Backup

This section will inform you of how to back up your files so you don't lose any data if something going wrong during the installation process.

1. First obtain a suitable storage device capable of storing the backup files on. It is recommended for the device to be around double the size of the size of the hard drive you want to back up. It is preferred to use an external hard drive as it has a lot of space and is portable but other storage devices can also be used, for example USB memory sticks, CDs and DVDs.
2. Plug the device in to your computer. Use a USB cable or other methods of connecting the device to the computer.
3. You can then use a backup wizard to complete the rest of the backup, but this will show you how to create a backup manually. First copy all the files of your system, then open up your external storage, then paste the files into the external storage. Then close the external storage and unplug it from the computer.
4. Store the backup in a safe place away from the computer so you can recover it if anything happens to the computer.
5. To restore the files from the backup, plug the external storage device containing the backup files, then copy them from the device and paste them into your computer at the hard drive location. These files will overwrite the current files and restore files that have been missing.

It is also useful to back up the original version of the game in case the current file gets deleted or corrupted. This way, you don't have to download the game again and you can just restore the copy of the game. Following the steps above will back up the game.

*(e) Evaluation***(i) Discussion of the degree of success in meeting the original objectives**

The discussion below shows the objectives that I planned to meet in the design sections. It includes whether they have been met or not.

Design Requirements

Requirement	Met?
The screen size will be 300 x 300 pixels	Yes, in ratio of 16:9, shown in screenshots
The colour of the background will be Black	Yes, shown in screenshots
The colour of all text will be White	Yes, shown in screenshots
The colour of the buttons will be Black	Yes, shown in screenshots
The player controlled sprite will be a white circle larger than 10 x 10 pixels	Yes, drawn as an image
The colour of the circle will be White	Yes, shown in screenshots
The good item will be a square smaller than the circle	Yes, drawn as an image
The colour of the good item will be green	Yes, shown in screenshots
The bad item will be a square smaller than the circle	Yes, drawn as an image
The colour of the bad item will be red	Yes, shown in screenshots
The power-up item will be a square smaller than the circle	Yes, drawn as an image
The colour of the power-up item will be Yellow	Yes, shown in screenshots

Input Requirements

Requirement	Met?
The player moves the circle by clicking on the screen where they want the circle to move to	Yes, shown in circle sprite movement tests
There should be one good and bad item on screen to start with but more than one of each item can be on the screen at any time as the game is played	Yes, shown in alpha testing during implementation

Processing Requirements

Requirement	Met?
It must be determined with each movement whether the circle interacts with a good or bad item, in which case the item will be randomly moved to another position on the game screen	Yes, can be seen from gameplay
If the circle interacts with a good item it should grow	Yes, can be seen from gameplay
Points should be increased by 100	Yes, shown in alpha testing during implementation

If the circle grows to a certain size the game should end and a “You win! :D” message be displayed	No, no evidence of this
If the circle interacts with a bad item it should shrink	Yes, can be seen from gameplay
Points should be decreased by 50	Yes, shown in alpha testing during implementation
If the circle shrinks to a certain size the game should end and a “You lose! :(” message be displayed	Yes, shown in alpha testing
The power-up item will appear at a certain time and position on the screen during gameplay	Yes, can be seen from gameplay
It must be determined with each movement whether the circle interacts with a power-up item, in which case the item will disappear and appear at another certain time and position on the game screen	Yes, can be seen from gameplay
If the circle interacts with a power-up item it should grow by more than the good item	Yes, can be seen from gameplay
Points should increase by 150	Yes, shown in alpha testing during implementation
The timer should start as soon as play is pressed	Yes, can be seen from gameplay
System must increment counter for the in game timer for every second that has elapsed	Yes, can be seen from gameplay
It should not end until the game is ended by the circle getting too big or too small	Yes, can be seen from gameplay
If it reaches one minute the game should end and a “Time’s up!” message should be displayed	Yes, shown in alpha testing during implementation

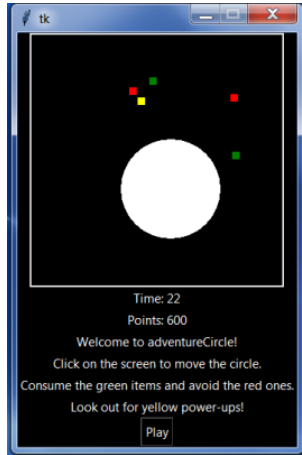
Output Requirements

Requirement	Met?
Current time to show the length of time played should be displayed at all times on screen	Yes, shown in screenshots
Current points should be displayed at all times on screen	Yes, shown in screenshots
The position of the circle will be displayed in real time	Yes, shown in screenshots
The position of all items will be displayed	Yes, shown in screenshots

The only requirement that was not met was the processing requirement:

- o **If the circle grows to a certain size the game should end and a “You win! :D” message be displayed**

The reason it was not met is that I did not have the expert knowledge required to achieve this outcome. It was found out that during my testing, instead of displaying “You win! :D”, the game simply carries on:



The code to win was originally:

```
elif 10*self.Size>600:
    self.TimerText['text']="You win! :D"
    self.end()
```

And to try and make it work I changed it to:

```
elif self.Size>=6:
    self.TimerText['text']="You win! :D"
    self.end()
```

It had no effect. Due to time and specialist knowledge limitations I was unable to properly fix this problem and meet the requirement fully.

(ii) **Evaluate the user's response to the system**

I asked my primary end user, my younger brother Ayo, to respond to the final version of my system and discuss his thoughts and ideas. The feedback received is shown below.

"I have played the final version of the game and I feel it was created to a very good standard and met almost all of the requirements. It is very good for children my age because it is very user and child friendly. The interface is easy to navigate and the game is easy to pick up straight away if never played before. To improve, I would suggest a level system that increases things like screen size and ball speed. To make it more challenging, the timer for the game to end should decrease for each level advanced to. Though there was a problem with the requirement to win the game, the game still ended when it was won so it wasn't that much of a problem. Overall my friends and I are happy with the game

Ayo"

The feedback has shown that my brother and the rest of the end users are very pleased with the finished game and it works fine without any bugs. My brother outlined possible extensions such as a level up system and changes in the dimensions, timer system and ball speed to go along with it. I think it is a good idea and something that can be looked into for future development. I am happy that the issue of the requirement not met was not a major one and it can be sorted with more time.

(iii) **Desirable extensions**

Overall, I think the game I created was successful and has accomplished the aim of the task. I have made a list of good, bad and extension points as part of my evaluation; this is shown below.

Good Points

- No errors or bugs occurred in implementation and alpha, beta and gamma testing
- End user group like the aesthetics of the game
- They found the functionality good and the game easy to play

Bad Points

- Not all requirements were met
 - To improve this limitation I would spend more time and do more research to find out how to solve the issue

Possible Extensions

- Add a level up system
 - To do this I would need to implement a max/highest points system where once that amount of points is reached, the next level is unlocked and all the extra features are applied
- Add a high score system
 - To do this first I would need to see if this is required by enough users because when first developing the system and coming up with the requirements this was something that was suggested by an end user but I discarded it and decided not to mention it because I didn't think it was significant enough. Now looking back in hindsight, seeing other games of this type all have a high score system and because it has been mentioned in the extensions, it is something to look into. To do this I would have to get the requirements from end users (whether first name or three letters, top ten or top five high scores etc.) and then code it into the game.
- Add additional and more complex power-ups
 - To do this I would have to do an investigation/analysis of what other power-ups would be relevant to my game, be feasible to implement and not clutter up the (already small) game screen. This, of course, would involve my end users and some input from them from other games possible. I would then begin programming again and add the required code needed for the additional power-ups

References/Bibliography

Figure 1:

<https://en.wikipedia.org/wiki/Pong>

Figure 2:

<http://www.businessinsider.com/a-mesmerizing-gif-of-a-perfect-game-of-snake-2013-4?IR=T>