Kiran DARJI

Highdown School and Sixth Form Centre

Bridge builder

Abstract

The objective of this project is to create a bridge building simulation, involving simple graphics, a database, object orientated principles and a physics engine. Languages used are SQL and Python

Contents

[Analysis 6](#_Toc8207609)

[Project outline 6](#_Toc8207610)

[Objective 6](#_Toc8207611)

[Target audience 6](#_Toc8207612)

[Research 6](#_Toc8207613)

[Programming language and physics libraries 6](#_Toc8207614)

[Maths and Physics 8](#_Toc8207615)

[Objectives 9](#_Toc8207616)

[Target audience interview 9](#_Toc8207617)

[SMART criteria – user needs 11](#_Toc8207618)

[Data flow diagrams 12](#_Toc8207619)

[Login 12](#_Toc8207620)

[Register 12](#_Toc8207621)

[Building 12](#_Toc8207622)

[Testing 12](#_Toc8207623)

[Acceptable limitation 13](#_Toc8207624)

[Design 14](#_Toc8207625)

[Hierarchy Diagram 14](#_Toc8207626)

[SMART 1 14](#_Toc8207627)

[SMART 2 15](#_Toc8207628)

[SMART 3 16](#_Toc8207629)

[SMART 4 17](#_Toc8207630)

[SMART 5 18](#_Toc8207631)

[PyBox2d 19](#_Toc8207632)

[GUI design 25](#_Toc8207633)

[Menus 25](#_Toc8207634)

[Login/Register 25](#_Toc8207635)

[Instructions 25](#_Toc8207636)

[Difficulty selector 26](#_Toc8207637)

[Build 26](#_Toc8207638)

[Bridge selector 26](#_Toc8207639)

[Test 27](#_Toc8207640)

[Entity relationship diagrams 28](#_Toc8207641)

[OOP classes 29](#_Toc8207642)

[TextBox 29](#_Toc8207643)

[InputBox 29](#_Toc8207644)

[Buttons 30](#_Toc8207645)

[StickButtons 31](#_Toc8207646)

[Materials 31](#_Toc8207647)

[Vehicles 34](#_Toc8207648)

[Class diagram - Materials 36](#_Toc8207649)

[Class diagram - Vehicles 37](#_Toc8207650)

[Class diagram - Button 38](#_Toc8207651)

[Pseudocode – important algorithms 39](#_Toc8207652)

[Build 39](#_Toc8207653)

[MAIN 47](#_Toc8207654)

[Save 55](#_Toc8207655)

[Test 57](#_Toc8207656)

[Classes 65](#_Toc8207657)

[Data Structures 68](#_Toc8207658)

[Stack 68](#_Toc8207659)

[Dictionary 68](#_Toc8207660)

[Adjacency List 68](#_Toc8207661)

[SQL Queries 69](#_Toc8207662)

[Python parameterised queries 69](#_Toc8207663)

[Creating the database 69](#_Toc8207664)

[Creating User table 69](#_Toc8207665)

[Creating Bridge table 70](#_Toc8207666)

[findUsername 70](#_Toc8207667)

[findEmail 70](#_Toc8207668)

[addUser 71](#_Toc8207669)

[findUser 71](#_Toc8207670)

[getUser\_ID 71](#_Toc8207671)

[findBridge 71](#_Toc8207672)

[addBridge 71](#_Toc8207673)

[getBridges 72](#_Toc8207674)

[deleteBridges 72](#_Toc8207675)

[getBridgeFile 72](#_Toc8207676)

[updateBridge 72](#_Toc8207677)

[getBridgeID 72](#_Toc8207678)

[Implementation – important algorithms 73](#_Toc8207679)

[File Structure 73](#_Toc8207680)

[Build 73](#_Toc8207681)

[loadBridge 73](#_Toc8207682)

[deleteExcessJoint 76](#_Toc8207683)

[Build Loop 77](#_Toc8207684)

[MAIN 82](#_Toc8207685)

[createButtonsLoad 82](#_Toc8207686)

[checkLogin 86](#_Toc8207687)

[checkRegister 88](#_Toc8207688)

[Main 91](#_Toc8207689)

[Save 92](#_Toc8207690)

[checkSaveName 92](#_Toc8207691)

[creating adjacencyList / findConnections 94](#_Toc8207692)

[Test 96](#_Toc8207693)

[createJoints 96](#_Toc8207694)

[loadBridge 99](#_Toc8207695)

[Main 102](#_Toc8207696)

[Classes 106](#_Toc8207697)

[Material.ifBreak 106](#_Toc8207698)

[Material.createBody 108](#_Toc8207699)

[Testing 111](#_Toc8207700)

[Evaluation 130](#_Toc8207701)

[SMART criteria completion 130](#_Toc8207702)

[User Feed Back 132](#_Toc8207703)

[Student 1 132](#_Toc8207704)

[Student 2 132](#_Toc8207705)

[Student 3 132](#_Toc8207706)

[Student 4 132](#_Toc8207707)

[Student 5 132](#_Toc8207708)

[Student 6 132](#_Toc8207709)

[Future Development 133](#_Toc8207710)

[GUI 133](#_Toc8207711)

[Online integration 133](#_Toc8207712)

[Performance 133](#_Toc8207713)

[General game ideas 133](#_Toc8207714)

[OOP design 133](#_Toc8207715)

[Appendix 135](#_Toc8207716)

[Testing screenshots 135](#_Toc8207717)

[Adjacency List Structure 171](#_Toc8207718)

[Example Bridge File 172](#_Toc8207719)

[References 173](#_Toc8207720)

[Source code 174](#_Toc8207721)

[MAIN.py 174](#_Toc8207722)

[Classes.py 189](#_Toc8207723)

[Build.py 206](#_Toc8207724)

[Test.py 212](#_Toc8207725)

[BridgeData.py 217](#_Toc8207726)

[Save.py 221](#_Toc8207727)

[Graphics.py 225](#_Toc8207728)

# Analysis

## Project outline

### Objective

For this project I have chosen to create a bridge building simulation in which a user can construct a bridge across a gap, the bridge will be tested by vehicles of varying weights to test its strength and integrity. This project is inspired by the game “Polybridge” by developers Dry Cactus, a game that involves giving the user obstructions and challenges to build their bridge on and around, ultimately to allow a car to get from one side of a river to another.



http://polybridge.drycactus.com/

### Target audience

The target audience for a program like this will be teenagers who have an interest in engineering and want to understand the basics of building a stable structure that will experience heavy weights; the maths involved in resolving forces within the bridge is similar to what students have to calculate in mechanics and physics. This program will also be usable by teachers who can present different problems or bridges to students

## Research

### Programming language and physics libraries

#### C++

C++ is a very suitable language for this project due to its physics library Box2D. The game this project is inspired by, “Polybridge” is built using Box2d; this library is commonly used, as is C++, this can allow for an easy distribution of the program once it is available. C++ was built as an extension to C to include object orientation, because these extensions were built onto a pre-existing language, this language is harder to understand and use. This project will rely heavily on object orientated principles thus making this issue a large negative of C++.

#### Python

Box2D has a Python binding called PyBox2D. This library provides the same functions as Box2D, however PyBox2D is not a widely used library and it is old, making it difficult to download. Python provides many other libraries, PyBox2D has an integrated test bed GUI, however the graphics are basic and there are many options to change the simulation, to encapsulate this I would need to use Pygame to create a bright and colourful GUI that hides the intricacies of PyBox2D. Python supports object orientation, the downside to using python OOP is that you cannot create interfaces however python supports multiple inheritance which can be used instead. Also all variables in python are publicly accessible, however I will design the algorithm as if variables and attributes were required to be passed.

#### JavaScript

JavaScript is primarily used for web applications, it is a viable choice as a language if this project were to be implemented online. Making the program accessible online is a potential objective, it would make it easier for users to access the program wherever they are. It is possible to implement object orientated programming principles within JavaScript however the inheritance is not implemented classically which will be required in this project.

For this project I have chosen to use Python, it is the language I am most familiar with, allowing me to focus on creating my algorithms rather than learning to use a language; it also has many libraries which I can use to my advantage. I shall also use MySQL to access and edit the database.

## Objectives

### Target audience interview

To help make my program specific to user needs I have interviewed physics students and teachers on what they would expect from such a program:

#### Student 1

“I want to see a variety of materials in the game to see how each one will affect the integrity of the structure. How the bridge will withstand against a range of heavy vehicles. I would like there to be clear instructions on how to use the game. I want the physics to be explained to me.”

#### Student 2

“I would like to be able to build a bridge and test it, and visually see the resolving of the forces.”

#### Student 3

“It would be useful if there were a clear user interface in which you can drag and drop objects. It would be beneficial to see different terrains and how the environment will affect the shape of the bridge.”

#### Student 4

“I would like to be able to crash things into the bridge, such as dropping boulders on it or firing projectiles, testing the integrity of a variety of materials in a selection of different environments.”

#### Student 5

“I would expect to see the material density their respective forces to be considered in the program. I would also like there to be comprehensible guidance within the program.”

#### Student 6

“I would expect to be able to see the effects that different vehicles will have on the bridge and how the bridge will react to these forces to give a good representation it would give in the real world.”

#### Student 7

“I would like to see an option to select what materials I wish to use to build, I want to be able to able change the colour of the materials.”

#### Teacher 1

“Often when trying to access materials online the internet fails, I would like it if such an application would be available offline.”

#### Teacher 2

“I would expect the ability to save bridges, I could set up an example before class, save it, then load during class. Via email or by other method I would like to share the bridges I have built or access other peoples bridges.”

#### Summary/User Needs

Most students expected a substantial amount of variation in the game, such as different vehicles or materials to use. They also expect to receive feedback on how the bridge has been affected what areas need to be strengthened. Some students commented on clear instructions for playing the games, and for the physics behind it to be explained. Teachers were mainly concerned about having access to the application in lesson time, it should be accessible offline and details about a bridge should be saved.

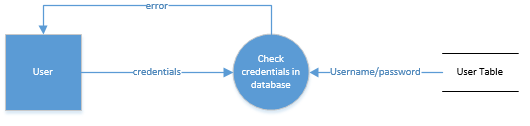
### SMART criteria – user needs

1. When the program is first turned on the user will be presented with a menu.
   1. The first option is to log on; the user will be asked to enter their username and password, if any details are incorrect the user will be alerted, and they will have to re-enter their details.
   2. The second option is to register; the user will be required to enter their name, email and desired username and password, the password will have a second box to check the password is correct. If the username or email is already taken or the passwords do not match the user will have to re-enter their details or alter them appropriately.
   3. The third option is to quit the game, this will simply leave stop the program from running.
2. Once logged on the user will be presented with a further 3 options.
   1. The first option named “Start” will take the user to a further menu (explained in Smart Criteria 3).
   2. The second option will show the user what the objective of the game is and how to play the game.
   3. The third option will log the user out taking them back to the first menu.
3. After the “Start” option is selected the user will be displayed with a further menu.
   1. The first option will be to create a new bridge: the user will be given a series of options to select the difficulty and shape of the land they will be building on.
   2. The second option will give users access to their previously built bridges so they can either edit them or test them.
   3. The third option will simply return the user to the previous menu.
4. The user will be able to build on the landscape they have chosen.
   1. The user will be able to select from a small variety of materials to build from, they will be able to select where they place their material on a grid
   2. The user will be able to undo the last build they made and delete any previous materials placed
   3. The user will also be able to save their bridge (giving it a name if it already doesn’t already have one) and click a test button that takes them directly to the testing screen with their bridge
5. The user will be able to test their bridge
   1. The user can choose from a small selection of vehicles to test their bridge with
   2. The vehicle will be user controlled
   3. If the bridge breaks the user can return to the build menu to edit their bridge

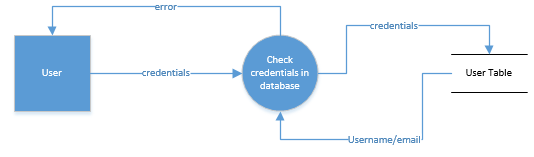
## Data flow diagrams

These data flow diagrams show when and where the database will be used

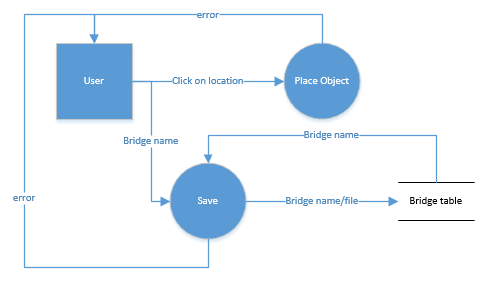
### Login



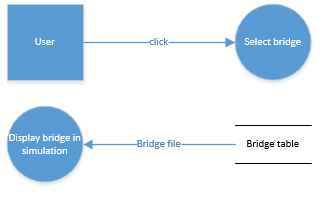
### Register



### Building



### Testing



## Acceptable limitation

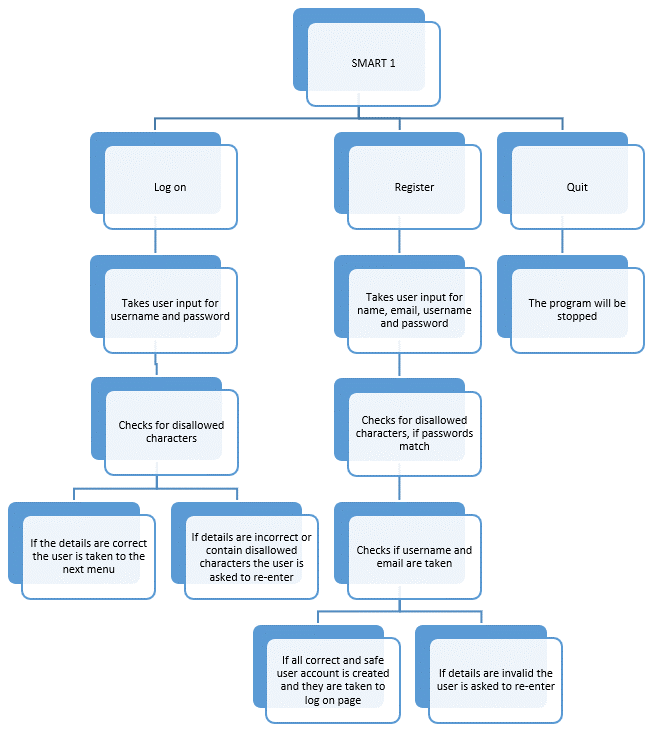
An acceptable limitation will be displaying the reaction force on a joint at any given time during testing, this is a part of the responsive feedback requested by some students.

# Design

## Hierarchy Diagram

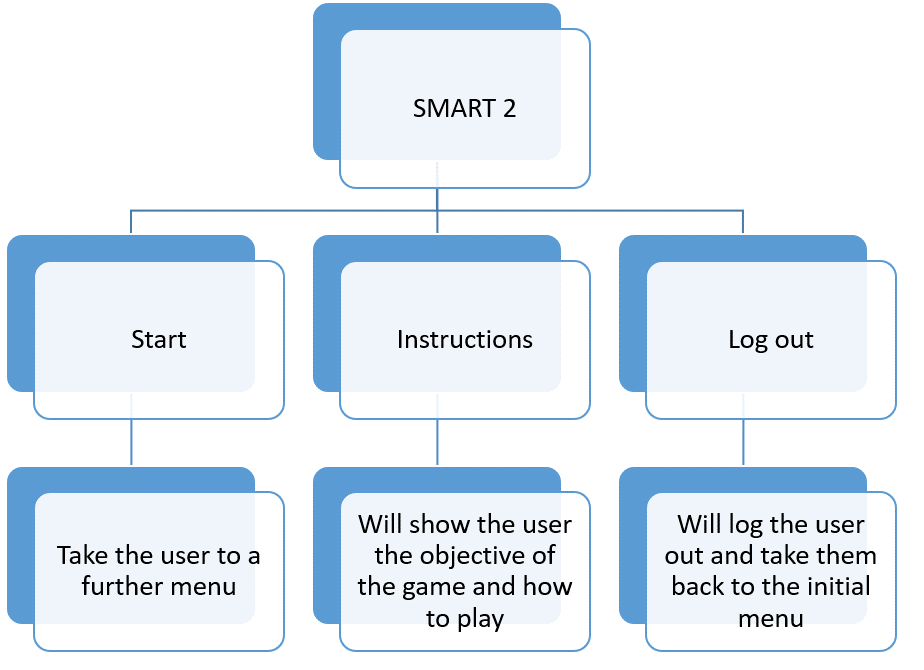
### SMART 1

This shows the breakdown of what the user first sees of the game, they will be able to login or register if they have not already got a login, the user can also quit the game if they choose ending the game.



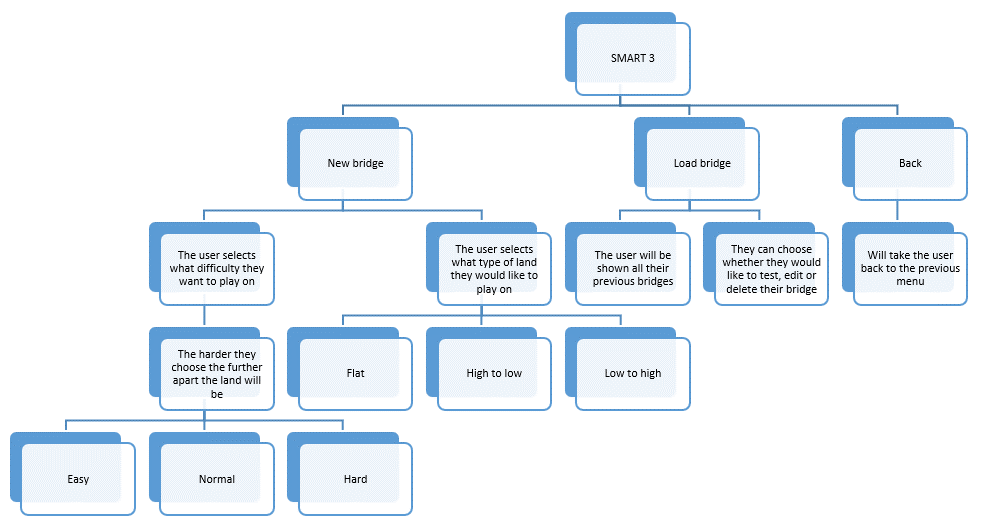
### SMART 2

This hierarchy diagram shows the breakdown of what the user sees after they login. Clicking start is expanded on the next hierarchy diagram and logging out takes the user back which is explained in the previous diagram.



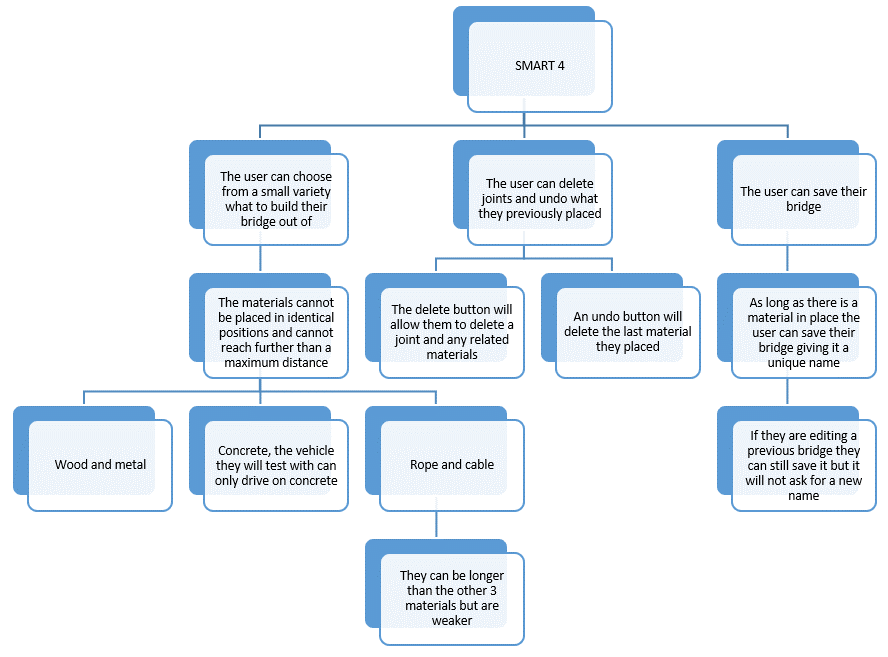
### SMART 3

This diagram shows a complex collection of menus. It shows the choices the user will be given when they decide to create a new bridge or load previous ones.



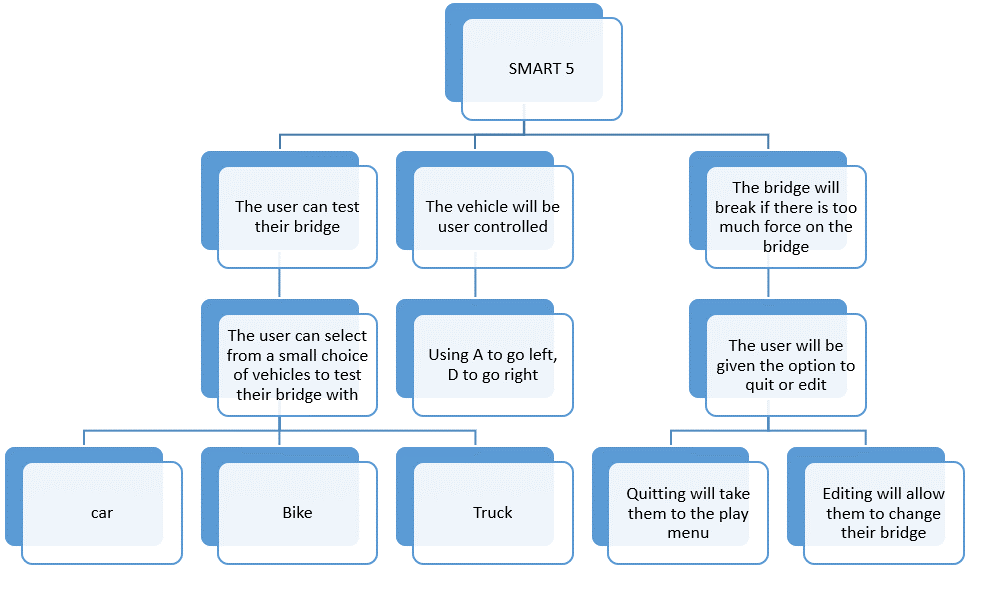
### SMART 4

This diagram shows what is available to the user whilst building their bridge, the choices for materials and other buttons available on screen.



### SMART 5

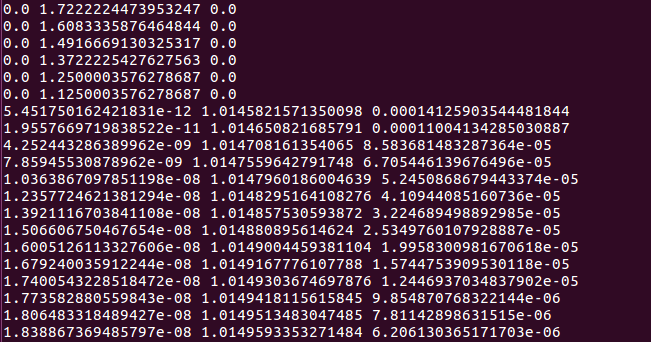
This diagram shows what the user can do to test their bridge, the small variety in vehicles and what will happen if their bridge is not good enough.



## PyBox2d

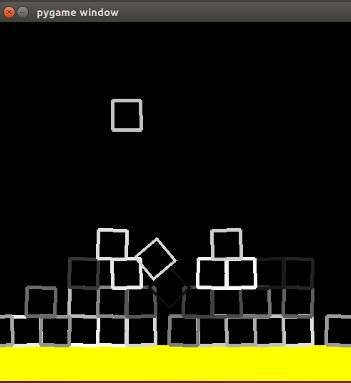
PyBox2D is a physics engine that’s allows for the simulation of a two-dimensional world that I will use in this project. The download package comes with a manual and examples on how to use Pybox2D which I learned from to use the engine.

The hello world example creates a 2x2m box and drops in onto a large static body, the program has no graphics and outputs the location and angle of the box:



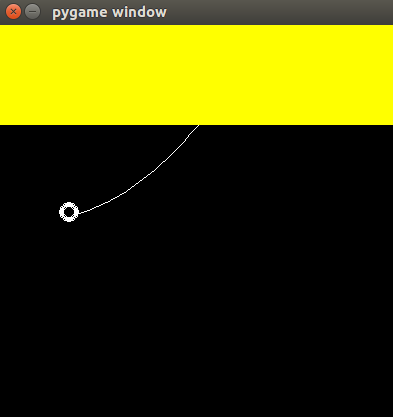
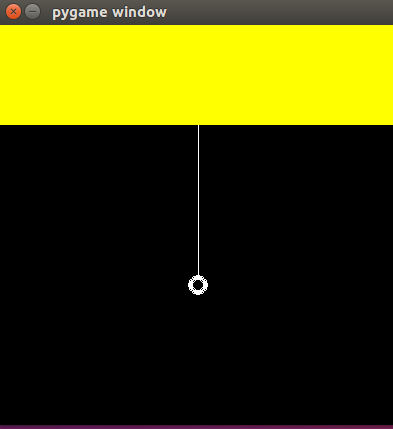
PyBox2D uses floating point arithmetic, the virtual box is virtually flat, but a negligible angle is still measured.

As a first task to learn PyBox2D I built off the hello world example. I altered the program to randomly drop boxes onto the static body, I also created a simple GUI to see how the boxes moved.

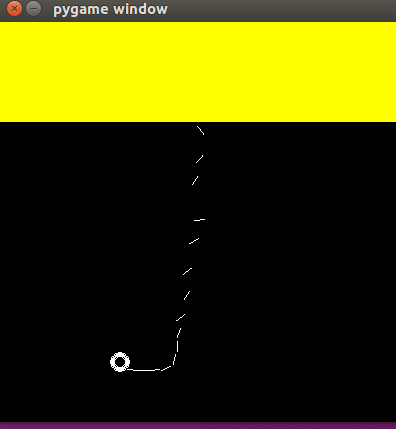


With each step of world simulation, PyBox2D performs a user specified amount of iterations to calculate the position and velocity of each body, the more iterations performed the more accurate the simulation will be. The manual recommends running ten iterations for both position and velocity, increasing the number of iterations will start to affect the performance of the program. Another challenge faced in creating this simple program was the way the boxes were produced on screen. Pygame does not allow you to rotate rect objects so instead I used trigonometric functions from python’s maths library to create 4 lines based on the angle of the object. However, when running the simulation again you would notice the boxes rotated in the wrong direction, this is because the coordinate origin for Pygame is in the top left and increases going downwards whilst PyBox2D the origin was in the bottom left increasing upwards. To fix this you had to minus the angle of the box from 2pi.

The second challenge I set myself was to create a dangling chain with a ball on the end. PyBox2D can only create solid objects, the way to simulate a chain is to create many solid objects connected by revolution joints, joints connect two bodies such they move relative to each other as desired. PyBox2D allows you to apply a force anywhere, I made the program such that using WASD will apply a force in the chosen direction, this allowed me to see how the chain moved:

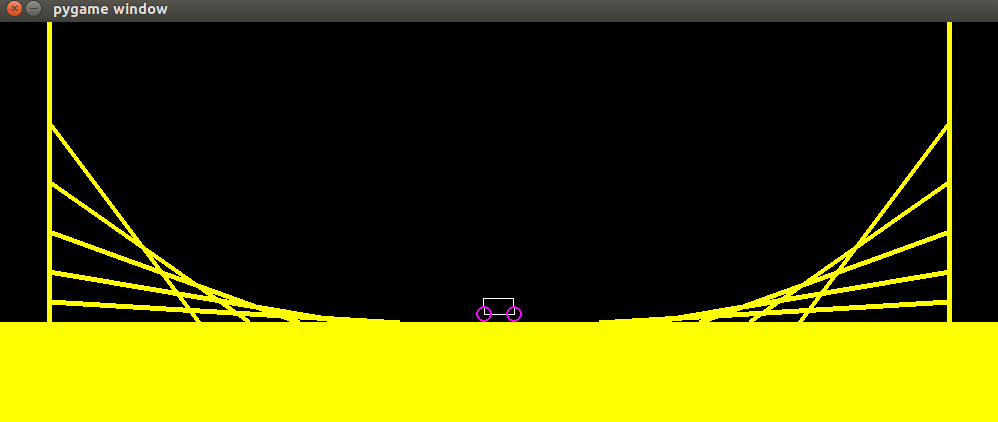


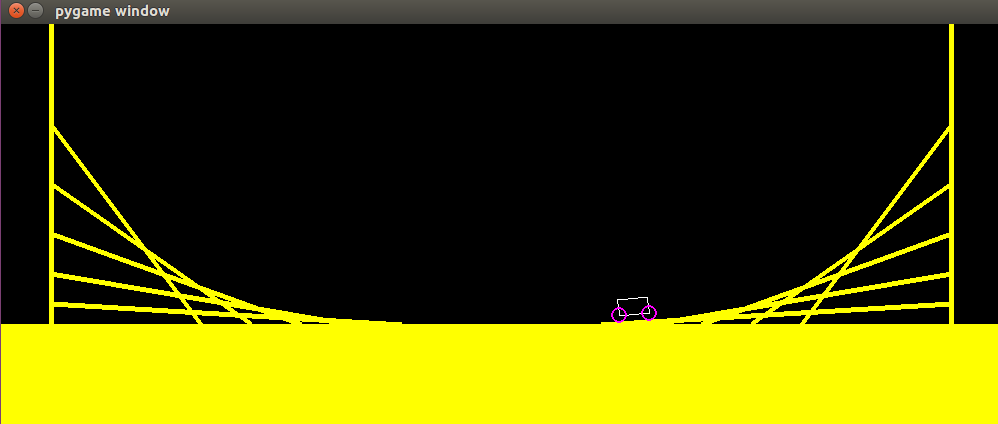
If a force to large for the program to simulate is applied the chain breaks and becomes permanently stuck:



For my program this force limit will not be reached, it won’t be a problem.

As a part of testing the user must be able to control a vehicle. There is an example of a simple car in the download package of PyBox2D, I used this example and gave it a simple GUI. The car consisted of three bodies, the chassis and 2 wheels. Joining the wheels to the chassis are two wheel joints; wheel joints are a combination of a revolute joint (allowing the wheel to spin) and a prism joint (simulating suspension), they have a motor speed which can be increased to turn the wheels:







PyBox2D has a function GetReactionForce() that is callable on joints, it returns the vector for the reaction force the joint applies to bodyB (the second body attached to the joint). This can be used to simulate the bridge breaking: if the magnitude of the force exceeds a certain limit the joint will break. I created a program that tested several bridges, either standalone or with a heavy box being dropped on it, the program outputs the reaction force of each joint, I took these results and plotted them using a spreadsheet.

The first bridge tested was a single body stretched across a gap:

The graph produced without a box:

With a box:

The joints experience a huge force when the box first hits the body, after the resting reaction force is higher than previously, this indicates this will be a good method for determining if the bridge will break. Also, the framerate does not seem to be affected by the large force:

The next bridge tested will be two bodies stretched across a gap, the results produced:

When the simulation started the materials dropped into position, sagging slightly. This is shown in the large initial peak. The bridge bounces up and down with “instability”, the variation in reaction force could be used in determining whether the bridge breaks, constant movement could weaken the bridge. When the simulation first starts to avoid the bridge breaking due to the large peak there will need to be a loading period to allow the program to simulate the bridge so it is in it resting position, the initial peak force does not occur in real life.

Dropping the box on the bridge shows the same trend of the joints experiencing a large spike in force. Due the bridge pulsating the box bounces up again explaining the decrease in force after the peak, once the box has settled on the bridge, the bridge continues to wobble.

Each joint can only connect to 2 pieces and return the reaction force of just one of the pieces, to get the total force a body experiences the body must be connected to every other body about that joint twice so every piece effects the other and so that the reaction force can be taken in each direction. The first of the following two graphs shows the results from a 7 body bridge, each piece only having two joints connected to it, the second graph shows the results from a 7 body bridge in which each piece is connected to every other piece about a joint twice. The magnitudes of each reaction force relating to a single piece are added together.

Both results exhibit the same shape, the bridge with more joints just produces larger forces, this is not because the bridge experiences a larger force but rather because more forces are being measured. Therefore, creating a joint between each piece about a joint in two directions is the more appropriate choice.

The manual for PyBox2D warns that the GetReactionForce() function is costly in terms of processing time:

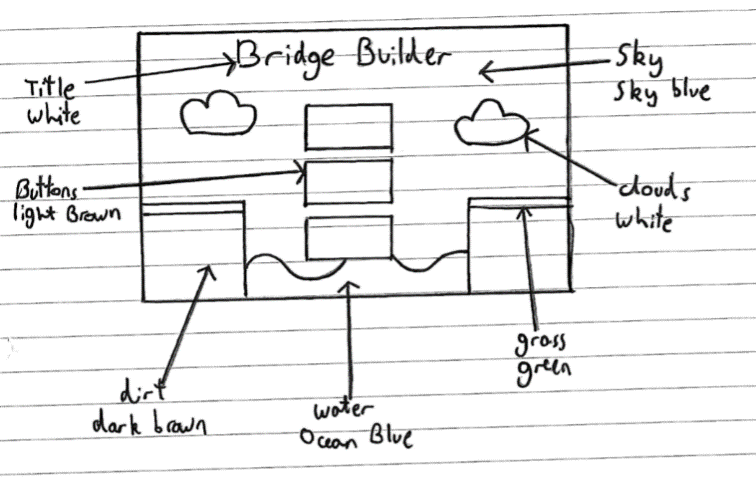
This graph comparing the framerate experienced in each simulation proves this. To combat this the calculation will only be performed once every 5 frames.

## GUI design

These are hand drawn designs for different sections of the GUI. The labels indicate what the object is and what colour it will be.

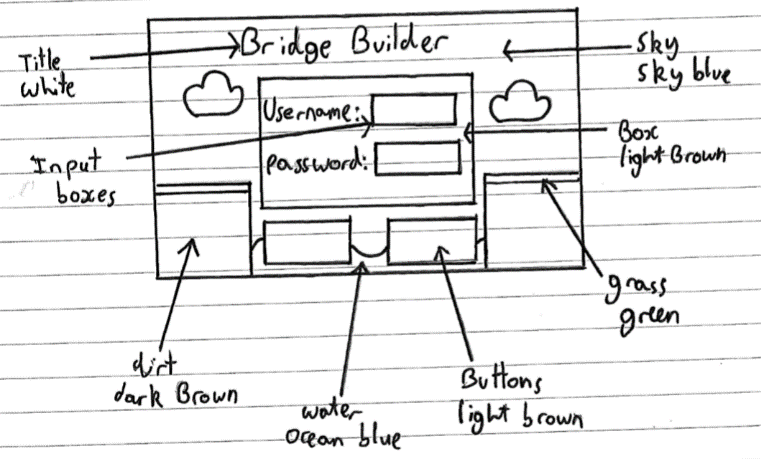
### Menus

All menus in the program that require a choice between three options will take this rough format. The three options will have buttons that will be in a single column in the middle of the screen.



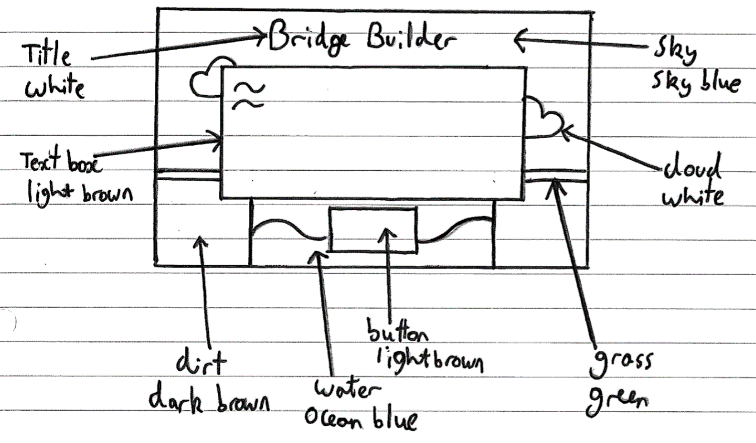
### Login/Register

Login and register will take the same rough form. There will be buttons login and back and input boxes for username and password when the user is logging in. Whilst registering there will be more input boxes for other required fields. The input boxes and their captions will be surrounded by a box.



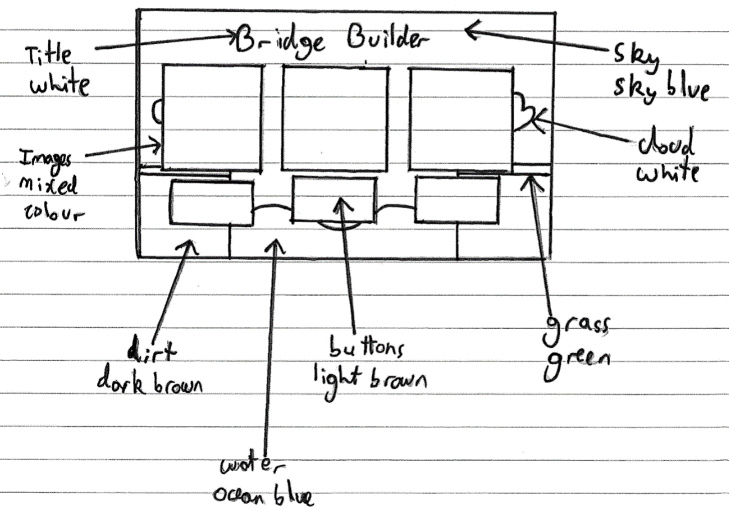
### Instructions

There will be a singular button called back and the instructions will be inside a box.



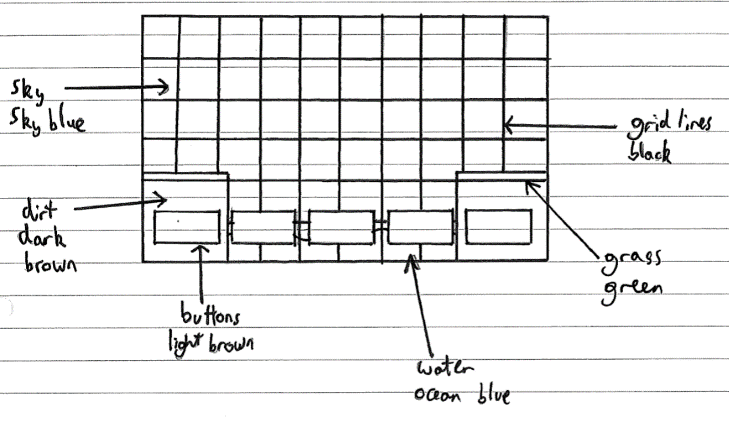
### Difficulty selector

The three images will show the type of terrain or difficulty to choose from, they each have a button below to be selected. There will also be a button below labelled back to take the user to the previous page.



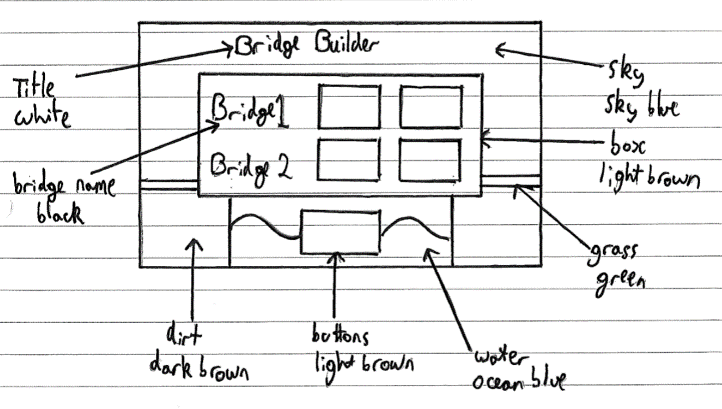
### Build

Buttons along the bottom allow the user to select what material they want to user. The grid shows them where they can place materials. There will also be bottoms in the top left for returning to the precious menu, saving and testing.



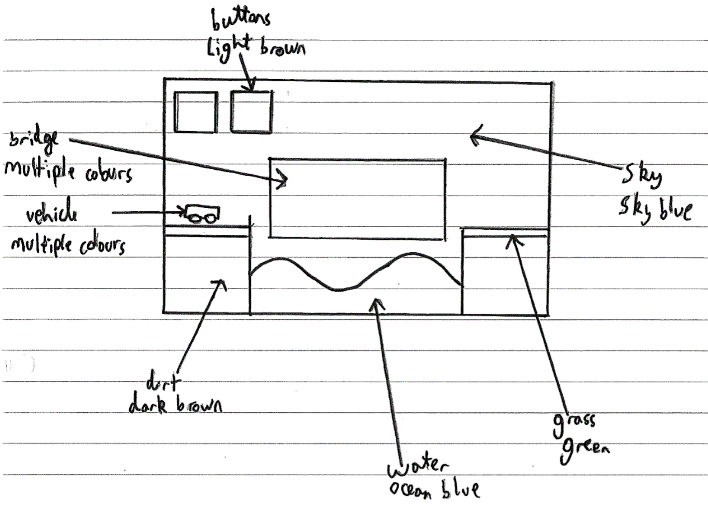
### Bridge selector

Buttons next to the bridge name will allow the user to choose from test, build and delete; these will all be inside of a light brown box. The button at the bottom of the screen will allow the user to return to the previous screen.



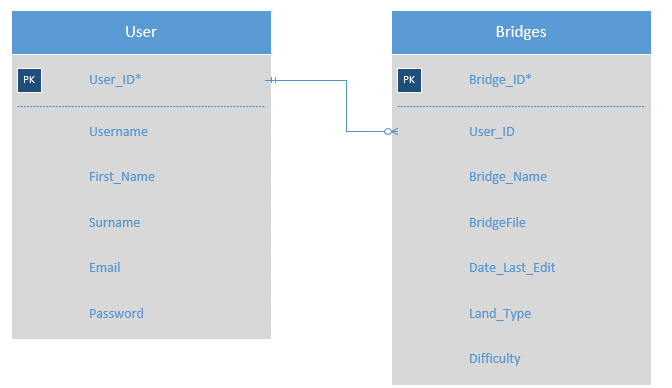
### Test

The bridge selected will appear in the central box. There will be buttons for returning to the previous screen and move to the section of the program to build the bridge.



## Entity relationship diagrams

This database will be simple with just 2 entities. One entity for holding user data and another for bridge data. The relationship is one user to many bridges.



\*- fields that can be auto-incremented by the database

With this method the BridgeFile attribute can be interpreted as none atomic, however this attribute only stores the file, the whole file is required at any one time so querying individual parts does not need to be worried about. An alternative method to remove this “problem” is to have separate entities for materials and joints that belong to a bridge, a downside to this is that the project will have a growing database element to it which was not the original objective.

## OOP classes

### TextBox

A text box is an element on a page, it is a brown octagonal box that will have text in its centre.

#### Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description/purpose** |
| Cap | String | To contain the message that will be displayed by the object |
| Height | Integer | The length of the object |
| Width | Integer | The width of the object |
| Centre | Coordinates (list of 2 integers) | Where the centre of the object will be |
| Colour1 | 3 integer tuple | The front colour of the text box |
| Colour2 | 3 integer tuple | The back colour of the text box |

#### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| create(window) | creates the text box in the location, size and text specified |

### InputBox

An input box is an element on a page that will allow the user to create text inputs.

#### Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description/purpose** |
| Cap | String | What the box will contain in game |
| Name | String | What the box will be captioned in game |
| xCo | Integer | The x coordinate of the box |
| yCo | Integer | The y coordinate of the box |
| Rect | Pygame rect object | Store the rect of the box |
| Active | Boolean | True if the user is currently typing into the box |

#### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| create() | creates the text box in the location, size and text specified |
| ifClick() | Determines whether the box has been clicked |
| Add() | Adds a character to the cap |
| makeActive() | Makes active true |
| deActive() | Makes active false |
| getActive() | Returns active |
| getCapLength() | Returns the length of cap |
| getCap() | Returns cap |

### Buttons

A button is an element on the page, the user can click on the button and the program will respond appropriately.

#### Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Description/ purpose** |
| Cap | String | To contain the word that will be on the button |
| Height | Integer | Height of the button |
| Width | Integer | Width of the button |
| Rect | List of 4 integers | Contains the coordinates and size of the objects pygame.rect |
| Locy | Integer | The y coordinate of the button |
| Locx | Integer | The x coordinate of the button |
| Colour1 | 3 integer tuple | The primary colour of the button |
| Colour2 | 3 integer tuple | The secondary colour of the tuple |

#### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| create(window) | Creates the button with the location and text specified |
| ifClick() | Using the rect variable it returns true or w=false whether the button has been clicked |

### StickButtons

Subclass of buttons. Similar to buttons except when it is clicked on the colour of the button stays dark until other buttons are clicked.

#### Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Description/ purpose** |
| On | Boolean | True if the button is on |

#### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| create(window) | Creates the button with the location and text specified |
| ifClick() | Using the rect variable it returns true or w=false whether the button has been clicked |
| getOn() | Returns On |
| turnOn() | Makes On true |
| turnOff() | Makes On false |

### Materials

Every material a user places is stored as a separate object described by this class. The subclasses describe the materials in more “detail” to allow the different materials to have different behaviours.

#### Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Description/ purpose** |
| x1 | Integer | The x coordinate of the first point of the material |
| y1 | Integer | The y coordinate of the first point of the material |
| x2 | Integer | The x coordinate of the second point of the material |
| y2 | Integer | The y coordinate of the second point of the material |
| Joint1 | Integer | Stores the index of the first joint the material is connected to |
| Joint2 | Integer | Stores the index of the second joint the material is connected to |
| Colour | 3 integer tuple | Stores the colour of the material |
| Body | Pybox2d body object | Stores the reference to the body |
| groupIndex | Integer | Determines what the material can collide with |
| Density | Integer | Stores the density of the material |
| B2joints1 | List | Stores the references to all the pybox2d joints at the first joint the material is connected to |
| B2joints2 | List | Stores the references to all the pybox2d joints at the second joint the material is connected to |
| maxForce | List | The maximum force the material can handle |
| Error | Boolean | True if the material cannot be placed where it is positioned |
| Material | String | Stores what material it is |

#### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| getMaxForce() | Returns the max force of the material |
| ifBreak() | Determines if the material will break |
| addB2joints1() | Adds a joint reference to B2joints1 |
| addB2joints2() | Adds a joint reference to B2joints2 |
| getBody() | Returns the reference to the pybox2d body for the material |
| getMaterial() | Returns material |
| errorTrue() | Makes error equal true |
| getJoint1() | Returns joint1 |
| getJoint2() | Returns joint2 |
| getError() | Returns error |
| getStart() | Gets the coordinates of the start of the material |
| getEnd() | Gets the coordinates of the end of the material |
| setJoint1() | Sets joint1 as the passed integer |
| setJoint2() | Sets joint2 as the passed integer |
| checkPlacement() | Determines whether the material can be placed where it is positioned |
| setCo1() | Sets the coordinates of start of the material |
| setCo2() | Sets the coordinates of the end of the material |
| updateLocation() | Changes the location of the end of the material |
| Draw() | Draws the material on screen |
| createBody() | Creates the pybox2d body for the material |
| testDraw() | Draws the material when in simulation |

#### Wood, steel and road

Both wood, steel and road classes are sub classes of material, they share the same methods and only update their attributes appropriately

#### Cable

A subclass of materials

##### Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Description/purpose** |
| Colour | List of 3 integers | Stores the RGB colour the material will have |
| Density | Integer | Stores the density of the object |

##### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| Draw() | produces the material on screen |
| updateLocation() | When in test mode this function will update the coordinates and rotation of the material according to calculations produced by pybox2d |
| testDraw() | Produces the material on screen in the testing stage |

#### Rope

A subclass of materials

##### Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Description/purpose** |
| Body | List | Stores the 10 bodies chained together simulating a rope |
| chainJoints | List | A list of joints holding the bodies together |

##### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| ifBreak() | Uses polymorphism to add onto the super’s function to determine whether one of the joints within the chain is broken |
| checkPlacement() | Overrides the super’s function to determine whether the material can be placed its current location |
| Draw() | Overrides the super’s function to draw a segmented line |
| createBody() | Creates the 10 pybox2d bodies for the material |
| testDraw() | Overrides the super’s function to draw the 10 bodies independently |

### Vehicles

Vehicles will be treated as objects, to allow for easy creation and manipulation.

#### Attributes

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Description/ purpose** |
| Chassis | Pybox2d body | Stores the reference to the body of the vehicle |
| Wheel1 | Pybox2d body | Stores the reference to the first wheel of the vehicle |
| Wheel2 | Pybox2d body | Stores the reference to the second wheel of the vehicle |
| Joint1 | PyBox2d joint | Stores the reference to the first joint of the vehicle |
| Joint2 | PyBox2d joint | Stores the reference to the first joint of the vehicle |

#### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| Forward() | Sets the joints motor speed to 30 |
| Back() | Sets the joints motor speed to -30 |
| Stop() | Sets the joints motor speed to 0 |

#### Car

Subclass of vehicle

##### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| Create() | Creates the bodies and joints to be used in simulation |
| Draw() | Draws the vehicle in simulation |

#### Bike

Subclass of vehicle

##### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| Create() | Creates the bodies and joints to be used in simulation |
| Draw() | Draws the vehicle in simulation |

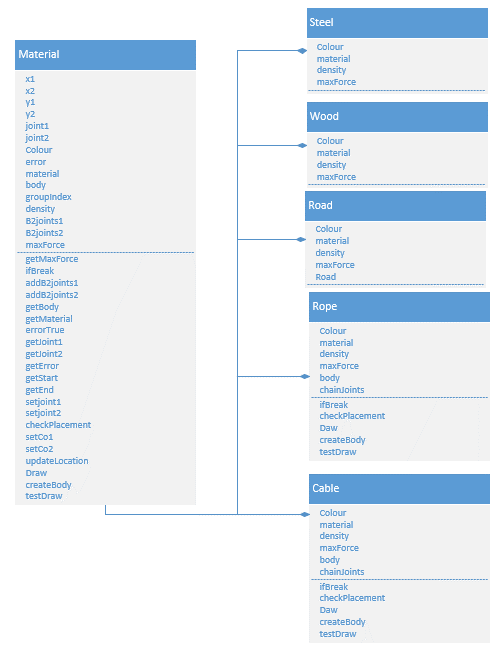
#### Truck

Subclass of vehicle

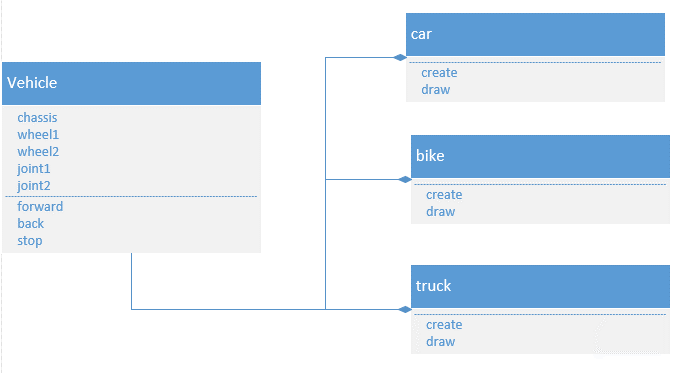
##### Methods

|  |  |
| --- | --- |
| **Name** | **Description/purpose** |
| Create() | Creates the bodies and joints to be used in simulation |
| Draw() | Draws the vehicle in simulation |

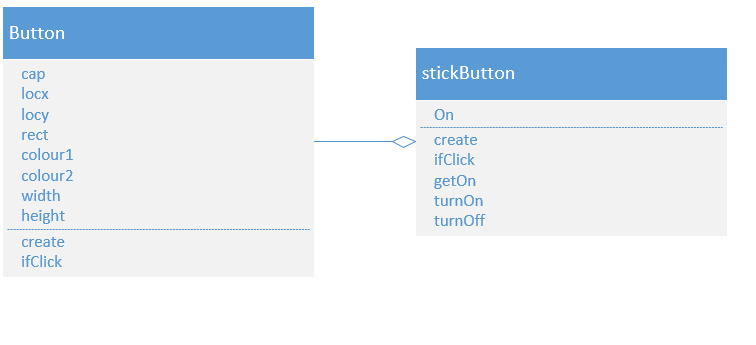
### Class diagram - Materials



### Class diagram - Vehicles



### Class diagram - Button



## Pseudocode – important algorithms

### Build

#### loadBridge

This subroutine extracts the bridge file from the database and breaks down the information in to several useful data structures and variables.

initiate data strucutres

jointList = LIST

materialStack = LIST

retireive data from database

bridgeFile, dif, land = GET BRIDGE FROM DATA BASE USING BRIDGE ID

jointNum = lenght(bridgeFile)

added = False

create list of joints

FOR joint = KEY IN bridgeFile

jointX,jointY = bridgeFile[joint]['location']

jointList.APPEND({'index':int(joint),'point':(jointX\*20,600 - jointY\*20)})

NEXT

Create list of materials

FOR joint = KEY IN bridgeFile

joint1 = joint

FOR connectedJoint = 0 TO LENGTH(bridgeFile[joint]['connectedJoints'])

joint2 = bridgeFile[joint]['connectedJoints'][connectedJoint]['joint']

Checks if material has already been added

FOR material = 0 TO LENGTH(materialStack)

IF (materialStack[material].getJoint1() = joint1

AND materialStack[material].getJoint2() = joint2)

OR (materialStack[material].getJoint1() = joint2

AND materialStack[material].getJoint2() = joint1) THEN

added = True

END IF

NEXT

If not added creates the material object

IF added = False THEN

IF bridgeFile[joint]['connectedJoints'][connectedJoint]['material'] = 'Steel' THEN

item = Classes.Steel()

ELSE IF bridgeFile[joint]['connectedJoints'][connectedJoint] ['material'] = 'Wood' THEN

item = Classes.Wood()

ELSE IF bridgeFile[joint]['connectedJoints'][connectedJoint] ['material'] = 'Road' THEN

item = Classes.Road()

ELSE IF bridgeFile[joint]['connectedJoints'][connectedJoint] ['material'] = 'Rope' THEN

item = Classes.Rope()

ELSE IF bridgeFile[joint]['connectedJoints'][connectedJoint] ['material'] = 'Cable'THEN

item = Classes.Cable()

END IF

Adds material to list and sets the joints appropriately

item.setjoint1(joint1)

item.setjoint2(joint2)

materialStack.APPEND(item)

ELSE

added = False

END IF

NEXT

NEXT

Set the coordinates of materials according to what joints they are connected to

FOR material = 0 TO LENGTH(materialStack)

FOR joint = 0 TO LENGTH(jointList)

IF materialStack[material].getJoint1() == jointList[joint]['index'] THEN

material.setCo1 = jointList[joint]['point']

ELSE IF materialStack[material].getJoint2() == jointList[joint]['index'] THEN

material.setCo2 = jointList[joint]['point']

END IF

NEXT

NEXT

#### deleteExcessJoint

when a material is deleted joints could be left behind, this subroutine deletes the joints

initiates list that will stores joints that need to be deleted

toDelete = List

FOR joint = 0 TO LENGHT(jointList)

numOfConnectedMaterials = 0

finds how many material are connected to current joint

FOR material = 0 TO LENGTH(materialStack)

IF jointList[joint]['index'] = material.getJoint1()

OR jointList[joint]['index'] = material.getJoint2()

THEN

numOfConnectedMaterials = numOfConnectedMaterials+ 1

END IF

NEXT

If a joint has no related materials it is added to the delete list

IF numOfConnectedMaterials = 0 THEN

toDelete.APPEND(jointList[joint])

END IF

NEXT

Deletes joints in delete list

FOR joint = 0 TO LENGTH(toDelete)

jointList.DELETE(toDelete(joint))

NEXT

#### Build Loop

This will be a part of the main subroutine in the Build file, this loop will handle user inputs.

WHILE 1 = 1 DO

Infinite loop until subroutine is exited

event = SYSTEM.EVENT

IF event = QUIT THEN

END

ELSE IF event = MOUSEDOWN THEN

click = True

ELSE IF event = MOUSEUP THEN

release = True

END IF

If the system senses the user either pressing down on the mouse, releasing it or quitting the program (clicking the red cross to close)

OUTUT BACKGROUND TO WINDOW

Calling a subroutine from graphics file

Material = WHICH stickButton IS ON

Calls function which returns which button is on

IF Material = "delete" THEN

clickStage = "Delete"

ELSE IF Material = "quit" THEN

clickStage = "Quit"

click = True

ELSE IF Material = "save" THEN

clickStage = "Save"

click = True

ELSE IF Material = "undo" THEN

clickStage = "Undo"

click = True

ELSE IF Material = "test" THEN

clickStage = "Test"

click = True

END IF

If button is not a material alter the “click stage” appropriately

FOR material = 0 TO LENGTH(materialStack)

materialStack[material].Draw()

NEXT

Draw all already placed materials, uses object orientation

FOR joint = 0 TO LENGTH(jointList)

DRAW JOINT AT jointList['point']

NEXT

Draw all already existing joints, uses subroutine from graphics file

PlaceMaterial1 is the initial click stage so all necessary buttons are shown on screen

IF clickStage = "PlaceMaterial1" THEN

FOR button = KEY IN stickButtons

stickButtons[button].create()

NEXT

PlaceMaterial2 is the secondary stage, this is after the user has chosen the first coordinate for a material. The material the user is currently placed will follow the mouse by updating the material according to where the cursor dot is, the cursor dot location is initialised in the next if statement but this doesn’t matter as it is virtually impossible for this else if to be triggered before the dot is initialised. This primarily uses the object methods.

ELSE IF clickStage = "PlaceMaterial2" THEN

item.updateLocation(dotX,dotY)

item.checkPlacement()

UPDATE item.error - WHETHER MATERIAL CAN BE PLACED IN CURRENT POSITION

item.Draw(window)

END IF

This if statement draws the cursor dot and update dotX and dotY as its location. The UI will have a grid across it and the dot will be placed on where the lines cross. This is achieved by the drawDot function, there will be 50 lines across the screen, this function will divide the pixel width by 50 and round it to the nearest integer and multiply it by 50 again, this will give a location that is directly on an intersection, the function then draws the dot in this location and returns the location.

IF Material <> "" THEN

IF Material = "delete" THEN

dotX,dotY = drawDot() RED

ELSE

dotX,dotY = drawDot() WHITE

END IF

IF click = True THEN

IF clickStage = "PlaceMaterial1" THEN

IF Material <> "" AND MOUSE NOT ON BUTTON THEN

IF Material = "steel" THEN

item = Classes.Steel()

ELSE IF Material = "wood" THEN

item = Classes.Wood()

ELSE IF Material = "road" THEN

item = Classes.Road()

ELSE IF Material = "rope" THEN

item = Classes.Rope()

ELSE IF Material = "cable" THEN

item = Classes.Cable()

END IF

If the user clicks on screen, is in the initial click stage and a material is selected, a material object is initialised

item.setCo1(dotX,dotY)

clickStage = "PlaceMaterial2"

The first coordinate of the material is initialised as the location of the cursor dot and the clock stage is moved on

jointPointAdded = False

FOR joint = 0 TO LENGTH(jointList)

IF (dotX,dotY) = jointList[joint]['point'] THEN

jointPointAdded = True

END IF

NEXT

IF jointPointAdded = False THEN

jointNum = jointNum + 1

jointDict = {'index':jointNum,'point':(dotX,dotY)}

jointList.APPEND(jointDict)

END IF

If in placing the first end of a material a new joint is created, this if statement and for loop checks if there is already a joint there, if not it creates one and appends it to the joint list

FOR joint = 0 TO LENGTH(jointList)

IF jointList[joint]['point'] = (dotX,dotY) THEN

item.setjoint1 (jointList[joint]['index'])

END IF

NEXT

This for loop checks through each joint and sets the material’s joint 1 to whichever is the matching joint

ELSE

TURN ON BUTTON THAT WAS CLICKED AND TURN OFF OTHERS

If the user clicks on a button rather than in a location for a material the button that is on is changed accordingly

END IF

ELSE IF clickStage = "PlaceMaterial2" THEN

IF item.getError() = False THEN

If the user clicks in a location during click stage 2 it means they want to place the second location for the material, it first checks whether the material can be placed in its current location, checking if the material is on top of another, whether the material is too long or too short

jointPointAdded = False

FOR joint = 0 TO LENGTH(jointList)

IF (dotX,dotY) = jointList[joint]['point'] THEN

jointPointAdded = True

END IF

NEXT

IF jointPointAdded = False THEN

jointNum = jointNum + 1

jointDict = {'index':jointNum,'point':(dotX,dotY)}

jointList.APPEND(jointDict)

END IF

FOR joint = 0 TO LENGTH(jointList)

IF jointList[joint]['point'] = (dotX,dotY) THEN

item.setjoint2(jointList[joint]['index'])

END IF

If a new joint is required a new joint is created and the joint 2 for the object is initialised

NEXT

materialStack.APPEND(item)

clickStage = "PlaceMaterial1"

The material is added to the material list and the click stage is moved back to the initial stage.

END IF

ELSE IF clickStage = "Delete" THEN

If the delete button is clicked the buttons disappear and the cursor turns red, this is handled by an earlier if statement

jointClicked = WHICH JOINT CLICKED(USE (dotX,dotY))

Function returns which joint was clicked on

jointCount = 0

cap = False

toDelete = List

IF jointClicked <> 0 THEN

FOR joint = 0 TO LENGTH(jointList)

IF jointList[joint]['index'] = jointClicked THEN

cap = True

END IF

IF cap = False THEN

jointCount = jointCount + 1

END IF

NEXT

DELETE jointList[jointCount]

Finds the joints index in the joint list and deletes it

FOR material = 0 TO LENGTH(materialStack)

IF materialStack[material].getJoint1() = jointClicked

OR materialStack[material].getJoint2() = jointClicked

THEN

toDelete.APPEND(material)

END IF

NEXT

FOR material = 0 TO LENGTH(toDelete)

DELETE materialStack[material]

NEXT

DELETE JOINTS WITH NO ASSOCIATED MATERIALS

Any materials related to the joint is also deleted, subroutine deletes any joints that no longer have any related materials

END IF

clickStage = "PlaceMaterial1"

TURN OFF BUTTONS

ELSE IF clickStage = "quit" THEN

info['build'] = False

RETURN ['sec',info]

If user clicks quit button the subroutine and file is exited

ELSE IF clickStage = "save" THEN

IF LENGTH(materialStack) <> 0 THEN

If materials exist save function is called

SAVE BRIDGE

END IF

clickStage = "PlaceMaterial1"

TURN OFF BUTTONS

ELSE IF clickStage = "undo" THEN

IF LENGTH(materialStack) <> 0 THEN

DELETE materialStack[-1]

END IF

Undo button deletes most recently added material

clickStage = "PlaceMaterial1"

TURN OFF BUTTONS

DELETE JOINTS WITH NO ASSOCIATED MATERIALS

ELSE IF clickStage = "Test" THEN

IF LENGTH(materialStack) <> 0 THEN

SAVE BRIDGE

info['build'] = False

RETURN ['chooseVehicle',info]

END IF

If the user chooses to test their brdge the subroutine and module is exited

clickStage = "PlaceMaterial1"

TURN OFF BUTTONS

END IF

click = False

UPDATE SCREEN

REPEAT

### MAIN

#### createButtonsLoad

This function creates the menu from which the user can select what bridge they want to edit, delete or test. The menu will be dynamic, there will be a maximum of 5 bridges to a page and there will be navigation buttons if a user has more. The current page the user is on will be a parameter and a returned variable that is altered outside of the function.

listOfBridge = List

listOfBridgeNames = List

listOfBridgeDates = List

buttonsOnScreen = 0

numOfScreens = 0

upButton = False

downButton = False

results = GET USER'S BRIDGES FROM DATABASE

loadMenu = CURRENT MENU NUMBER

start = loadMenu\*5

end = start + LENGTH(results) MOD 5

Variables are initialised, user bridges are retrieved from database. loadMenu us the current menu page where 0 is the first page. start is the index of the first bridge on page in the results list, calculated by multiplying which page the user is on by 5 (because the maximum bridges per page is 5). end is the index of the final bridge onscreen in the results list, calculated by adding the remainder of the number bridges when divided by 5 to the start.

IF LENGTH(results) <> 0 THEN

IF LENGTH(results) MOD 5 <> 0 THEN

If there is NOT a maximum number of bridges on screen, this distinction is made to determine how many pages there are in the menu. The DIV function will calculate the number of pages that have 5 bridges on them, if the number of bridges is not divisible by 5 exactly, numOfScreens must be incremented by 1 to account for the page that does not have maximum bridges on it

numOfScreens = LENGTH(results) DIV 5 + 1

IF loadMenu = numOfScreens - 1 THEN

If user is currently on final page, this distinction is made to determine what the boundaries are for the for loop, if the user is not on the final page it is simply the start element up to the 5 element after it, otherwise the end variable will be used as the up boundary

FOR bridge = start TO end

listOfBridgeID.APPEND(results[bridge][0])

listOfBridgeNames.APPEND(results[bridge][1])

listOfBridgeDates.APPEND(results[bridge][2])

buttonsOnScreen += 1

NEXT

ELSE

If user is not on final page

FOR bridge = start TO start + 5

listOfBridgeID.APPEND(results[bridge][0])

listOfBridgeNames.APPEND(results[bridge][1])

listOfBridgeDates.APPEND(results[bridge][2])

buttonsOnScreen += 1

NEXT

END IF

ELSE

If there is a maximum number of bridges on screen

numOfScreens = len(results) DIV 5

FOR bridge = start TO start + 5

listOfBridgeID.APPEND(results[bridge][0])

listOfBridgeNames.APPEND(results[bridge][1])

listOfBridgeDates.APPEND(results[bridge][2])

buttonsOnScreen += 1

NEXT

END IF

ELSE

OUTPUT "you have no bridges"

END IF

The three lists all have the same length varying from 1 to 5 depending on how many bridges are on screen. The names will be displayed so the user can identify them uniquely, a single element of dates will be used after the user has chosen a bridge, similarly only single element from the ID list will be used after the user has chosen a bridge

IF buttonsOnScreen >= 1 THEN

buttons['bridge1'].create()

END IF

IF buttonsOnScreen >= 2 THEN

buttons['bridge2'].()

END IF

IF buttonsOnScreen >= 3 THEN

buttons['bridge3'].create()

END IF

IF buttonsOnScreen >= 4 THEN

buttons['bridge4'].create()

END IF

IF buttonsOnScreen = 5 THEN

buttons['bridge5'].create()

END IF

If there is less than 5 buttons on screen than only some buttons will be shown on screen

IF loadMenu <> 0 THEN

buttons['up'].create()

upButton = True

END IF

If user is not on first screen create up button

IF loadMenu < numOfScreens - 1 THEN

buttons['down'].create()

downButton = True

END IF

If user is not on last page create down button

buttons['back5'].create()

OUTPUT listOfBridgeNames

RETURN [listOfBridgeID,listOfBridgeNames,listOfBridgeDates,buttonsOnScreen,upButton,downButton]

Returns required variables

#### checkLogin

This function checks whether a user’s login attempt is valid, it returns an error if not.

checkList = [" ",";","=","'","""]

Initialises a list of disallowed characters

error = List

list of errors

User\_ID = ""

emptyCount = 0

number of empty input boxes

invalidCharCount = 0

numbe of boxes containing invalid characters

FOR box = KEY in inputboxes

IF box = "Lusername" or box = "Lpassword" THEN

FOR char = 0 TO LENGTH(checkList)

IF inputBoxes[box].getCap() = "" THEN

emptyCount = emptyCount + 1

ELSE IF checkList[char] IN inputBoxes[box].getCap() THEN

invalidCharCount = invalidCharCount + 1

END IF

Checks if any required input boxes are empty or contain invalid characters

NEXT

END IF

NEXT

IF emptyCount > 0 THEN

error.APPEND("All fields must be entered")

if field is empty error is added to list

END IF

IF invalidCharCount > 0 THEN

error.APPEND("incorrect details")

if field contains invalid character error is added to list

END IF

IF LENGTH(error) = 0 THEN

username = inputboxes['Lusername'].getCap()

password = inputboxes['Lpassword'].getCap()

connect = COMPARE WITH DATABASE

if there are no errors the inputted data is compared to the database

IF connect = "Not" THEN

error.APPEND("Incorrect details")

if details do not match database error is added to list

ELSE IF connect = "Error" THEN

error.APPEND("Error connecting to database")

if there is an error retrieving data then error is added to list

ELSE

User\_ID = GET USER ID FROM DATABASE

If there are no errors the user’s ID is retrieved from the database

IF User\_ID = "" THEN

error.APPEND("Error connecting to the database")

if an error occurs whilst connecting tot eh database error is added to the list

END IF

END IF

END IF

RETURN error,User\_ID

#### checkRegister

similar to checkLogin however there are a few more checks required and the user login is created f input are valid and not duplcates.

checkList = [" ", ";", "=", "'", '"']

error = List

emptyCount = 0

FOR box = KEY IN inputboxes

IF (box <> "Lusername") AND (box <> "Lpassword")

FOR char = 0 TO LENGTH(checkList)

IF (inputboxes[box].getCap() = "") OR (checkList[char] IN inputboxes[box].getCap()) THEN

emptyCount = emptyCount + 1

END IF

NEXT

END IF

NEXT

IF emptyCount > 0 THEN

error.APPEND("All fields must be entered and cannot contain ;, =, ' or "")

END IF

Same as checkLogin, if any fields are null or contain invalid characters an error is added to list

IF (NOT("@" IN inputboxes['email'].getCap())) AND (inputboxes['email'].getCap() <> "") THEN

error.APPEND("Please enter a valid email")

END IF

A simple check to see if the email I valid, if the field does not contain “@” then an error is added to list

IF inputboxes['Rpassword'].getCap() <> inputboxes['RpasswordC'].getCap() THEN

error.APPEND("Passwords do not match")

END IF

User is required to enter password twice, if they do not match it is not a valid input and an error is added to list

IF error IS EMPTY THEN

username = inputboxes['Rusername'].getCap()

UsernameState = COMPARE USERNAME TO DATABASE

IF UsernameState <> "" THEN

IF UsernameState = "Taken" THEN

error.APPEND("Username taken")

ELSE IF UsernameState = "Error" THEN

error.APPEND("Error connecting to database")

END IF

END IF

Checks if chosen username is already taken, if it is then an error is added to list

email = inputboxes['email'].getCap()

EmailState = COMPARE EMAIL TO DATABASE

IF EmailState <> "" THEN

IF EmailState = "Taken" THEN

error.APPEND("Email taken")

ELSE IF EmailState = "Error" THEN

error.APPEND("Error connecting to database")

END IF

Checks if email is already used by another account, if it is then an error is added to list

END IF

END IF

IF LENGTH(error) = 0:

first = inputboxes['first'].getCap()

sec = inputboxes['sec'].getCap()

email = inputboxes['email'].getCap()

username = inputboxes['Rusername'].getCap()

password = HASH(inputboxes['Rpassword'].getCap())

connect = UPLOAD DATA TO DATABASE

if there are no errors the user details are uploaded to the database, password is hashed

IF connect <> "" THEN

error.APPEND("Error connecting to database")

if there is an error in uploading the data an error is added to list

END IF

END IF

RETURN error

Returns list of errors

#### Main

This is the first subroutine called, it sorts which modules is to be used next, either the main menu module, the build module or the test module

INITIALISE WINDOW

Next = "initial"

info = {'User\_ID':0, 'Bridge\_ID':0, 'build':False, 'test':False}

initialises necessary infomation

WHILE 1=1 DO

IF info['build'] = False AND info['test'] = False THEN

info = menuLoop(Next,window,info)

if a bridge is not being tested or built than the menu module is started

END IF

IF info['build'] = True THEN

Next,info = Build.Main(info,window)

END IF

IF info['test'] = True THEN

Next,info = Test.Main(info,window)

END IF

REPEAT

### Save

#### checkSaveName

Again checkSaveName is similar to checkRegister and checkLogin just with one input

checkList = [" ", ";", "=", "'", '"']

error = List

emptyCount = 0

bridgeID = ""

FOR box = KEY IN inputboxes

FOR char = 0 TO LENGTH(checkList)

IF (inputboxes[box].getCap() = "") OR (checklist(char) in inputboxes [box].getCap()) THEN

emptyCount = emptyCount + 1

END IF

NEXT

NEXT

IF emptyCount > 0 THEN

error.APPEND("All fields must be entered and cannot contain ;, =, ' or "")

END IF

Input box is checked to see if it null or contains any invalid characters, if there is then an error is added to list

IF LENGTH(error) = 0 THEN

name = inputboxes['name'].getCap()

nameState = COMPARE WITH DATABASE

if there are no errors then the name is compared to the database to see if it already taken, if it is then and error is added to the list

IF nameState = "Taken" THEN

error.APPEND("You have used this name already")

ELSE IF nameState = "Error" THEN

error.APPEND("Error connecting to database")

END IF

END IF

IF LENGTH(error) = 0 THEN

fileName = (name+"\_"+info['User\_ID'])

CREATE file - fileName

WRITE TO file - adjacencyList

connect = SAVE file TO DATABASE

if there is no errors the bridge is saved to the database, adjacencyList is converted to file to be stored in database.

DELETE file

IF connect <> "" THEN

error.APPEND("Error connecting to database")

ELSE

bridgeID = GET BRIDGE ID FROM DATABASE

END IF

END IF

RETURN [error,bridgeID]

#### creating adjacencyList

this algorithm is a part of a subroutine for saving the bridge, from the material list and joint list this algorithm and findConnections creates an adjacency list for the bridge explained in more detail in Data Structures.

adjacencyList = {}

initialise dictionary

FOR joint = 0 TO LENGTH(jointList)

For each joint

index = jointList[joint]['index']

joint index

x,y = jointList[joint]['point']

location

x = x/20

y = 30 - y/20

convert location from pixels to meters

location = (x,y)

connectedJoints = findConnections(index,materialStack)

calls function that returns list of connected joints and what material the connection is made with

adjacencyList[index] = {'location':location,'connectedJoints':connectedJoints}

add to adjacency list

#### findConnections

connectedTo = []

FOR material = 0 TO LENGTH(materials)

For each material

IF index = material.getJoint1() THEN

connectedTo.APPEND({'joint':material.getJoint2(), 'material':material.getMaterial()})

ELSE IF index = material.getJoint2() THEN

connectedTo.APPEND({'joint':material.getJoint1(), 'material':material.getMaterial()})

END IF

If current joint is connected to material add the material’s other joint as a connected joint

NEXT

RETURN connectedTo

### Test

#### createJoints

for each joint that exists this algorithm is ran

FOR materialA = 0 TO LENGTH(joint['materials'])

FOR materialB = 0 TO LENGTH(joint['materials'])

each material connects to every other material about a point in two directions

IF materialA <> materialB THEN

IF joint['materials'][materialA].getMaterial() = "Rope"

OR joint['materials'][materialA].getMaterial() = "Cable"

THEN

IF joint['index'] = joint['materials'][materialA].getJoint1() THEN

bodyA = joint['materials'][materialA].getBody()[0]

ELSE IF joint['index'] = joint['materials'][materialA].getJoint2() THEN

bodyA = joint['materials'][materialA].getBody()[-1]

END IF

ELSE

bodyA = joint['materials'][materialA].getBody()

END IF

rope and cable bodies consist of a list of 10 bodies

depending on whether the joint is the material's joint1 or joint2 the PyBox2D joint must

connect to either the first or last body

IF joint['materials'][materialB].getMaterial() = "Rope"

OR joint['materials'][materialB].getMaterial() = "Cable"

THEN

IF joint['index'] = joint['materials'][materialB].getJoint1() THEN

bodyB = joint['materials'][materialB].getBody()[0]

ELSE IF joint['index'] = joint['materials'][materialA].getJoint2() THEN

bodyB = joint['materials'][materialB].getBody()[-1]

END IF

ELSE

bodyB = joint['materials'][materialB].getBody()

END IF

Joint = CREATE REVOLUTE JOINT AT joint BETWEEN bodyA and bodyB

creates revolute joint at the position given by joint['point']

final parameter allows the materials to collide

IF joint['index'] = materialB.getJoint1() THEN materialB.addB2joints1({"reference":Joint,"maxForce":materialA.getMaxForce()})

ELSE IF joint['index'] = materialB.getJoint2() THEN

materialB.addB2joints2({"reference":Joint,"maxForce":materialA.getMaxForce()})

END

adds joint to materialB's joint list, used in calculating the reaction force

END IF

NEXT

NEXT

IF joint['onGround1'] = True THEN

bodyA = groundBody1

ELSE IF joint['onGround2'] = True THEN

bodyA = groundBody2

END IF

checks if the joint is connected to the ground

IF joint['onGround1'] = True OR joint['onGround2'] THEN

FOR material = 0 TO LENGTH(joint['materials'])

IF joint['materials'][material] = "Rope"

OR joint['materials'][material] = "Cable"

THEN

IF joint['index'] = joint['materials'][material].getJoint1() THEN

bodyB = joint['materials'][material].getBody()[0]

ELSE IF joint['index'] = joint['materials'][material].getJoint2() THEN

bodyB = joint['materials'][material].getBody()[-1] END IF

ELSE

bodyB = joint['materials'][material].getBody()

END IF

Joint = CREATE REVOLUTE JOINT AT joint BETWEEN bodyA and bodyB

IF joint['index'] = material.getJoint1() THEN

material.addB2joints1({"reference":Joint,"maxForce":material.getMaxForce()})

ELSE IF joint['index'] = material.getJoint2() THEN

material.addB2joints2({"reference":Joint,"maxForce":material.getMaxForce()})

END IF

NEXT

END IF

connects each material about that joint to the ground

#### loadBridge

materialStack,jointList,jointNum,dif,land = Build.loadBridge(bridgeID)

Dirt1Y,Dirt1width,Dirt1height,Dirt2X,Dirt2Y,Dirt2width,Dirt2height = Graphics.dirtSize(dif,land)

uses these variables to determine whether a joint is connected to the ground and to create the ground

Dirt1Y = 30 - Dirt1Y/20

Dirt1width = Dirt1width/20

Dirt1height = Dirt1height/20

Dirt2X = Dirt2X/20

Dirt2Y = 30 - Dirt2Y/20

Dirt2width = Dirt2width/20

Dirt2height = Dirt2height/20

converts to meters

FOR joint = 0 TO LENGTH(jointList)

convx,convy = jointList[joint]['point']

jointList[joint]['point'] = (convx,convy)

NEXT

converts each joint to meters

groundBody1 = CREATE GROUND BODY 1

groundBody2 = CREATE GROUND BODY 2

CREATE BOUNDARY WALLS

creates two ground bodies, the left and right island, and walls

FOR material = 0 TO LENGTH(materialStack)

materialStack[material].createBody()

NEXT

creates body for each material

FOR joint = 0 TO jointList

materialAboutJoint = []

FOR material = 0 TO LENGTH(materialStack)

IF jointList[joint]['index'] = material.getJoint1() OR jointList[joint]['index'] = material.getJoint2() THEN

materialAboutJoint.APPEND(materialStack[material])

END IF

jointList[joint]['materials'] = materialAboutJoint

NEXT

determines what materials connect to current joint

jointx,jointy = jointList[joint]['point']

IF (jointx = Dirt1width AND jointy <= Dirt1Y)

OR (jointx <= Dirt1width AND jointy == Dirt1Y)

THEN

joint['onGround1'] = True

joint['onGround2'] = False

ELSE IF (jointx = Dirt2X AND jointy <= Dirt2Y)

OR (jointx >= Dirt2X AND jointy == Dirt2Y)

THEN

joint['onGround1'] = False

joint['onGround2'] = True

ELSE

joint['onGround1'] = False

joint['onGround2'] = False

END IF

determines whether joint is connected to the ground

createJoints()

NEXT

RETURN (materialStack,jointList,dif,land)

#### Main

INITIATE WORLD

materialStack,jointList,dif,land = loadBridge(info['bridgeID'],world)

frameCount = 0

creates bridge

buttons = {

'quit':Classes.Button('Quit',150,50),

'edit':Classes.Button('Edit',400,50)

}

creates buttons

IF info['vehicle'] = "car" THEN

vehicle = Classes.car()

ELSE IF info['vehicle'] = "bike" THEN

vehicle = Classes.bike()

ELSE IF info['vehicle'] = "truck" THEN

vehicle = Classes.truck()

END IF

defines vehicle with object orientation

IF land = 1 THEN

height = 10

ELSE IF land = 2

height = 13

ELSE IF land = 3

height = 8

END IF

determines at what height the vehicles will be created

vehicle.create(height)

creates vehicle

forward,back,click = False,False,False

WHILE 1 = 1 DO

event = SYSTEM.EVENT

IF event = QUIT THEN

END

ELSE IF event = MOUSEDOWN THEN

click = True

ELSE IF event = KEYDOWN THEN

IF event.key = d THEN

IF back = False THEN

forward = True

END IF

ELSE IF event.key = a THEN

IF forward = False THEN

back = True

END IF

END IF

ELSE IF event = KEYUP THEN

IF event.key = d THEN

forward = False

ELSE IF event.key = a THEN

back = False

END IF

END IF

Depending on what key is pressed the car will move

IF frameCount MOD 5 = 0 AND frameCount > 100 THEN

FOR material = 0 TO LENGTH(materialStack)

materialStack[material].ifBreak()

NEXT

END IF

if not loading determine whether material will break

functions within ifBreak are costly in processing power only done once every 5 steps

OUTPUT GRAPHICS

FOR material = 0 TO LENGTH(materialStack)

materialStack[material].testDraw()

NEXT

vehicle.Draw()

IF frameCount > 100 THEN

IF forward THEN

vehicle.forward()

ELSE IF back THEN

vehicle.back()

ELSE IF (forward = False AND back <> True)

OR (back = False AND forward <> True) THEN

vehicle.stop()

END IF

ELSE

OUTPUT "Loading"

frameCount = frameCount + 1

END IF

If not loading allow the user to control vehicle otherwise show “loading” on screen

buttons['quit'].create(window)

buttons['edit'].create(window)

IF click = True THEN

IF buttons['quit'].ifClick() = True THEN

info['test'] = False

RETURN ["sec",info]

If user clicks quit button the subroutine and module is exited

ELSE IF buttons['edit'].ifClick() = True THEN

info['test'] = False

info['loadBridge'] = True

info['build'] = True

RETURN ["build",info]

If user edit quit button the subroutine and module is exited

END IF

END IF

WORLD STEP

UPDATE GRAPHICS

WAIT(5)

### Classes

#### Material.ifBreak

totalX = 0

totalY = 0

numOfMaterials = LENGTH(self.B2joints1) + LENGTH(self.B2joints2)

maxForce = 0

FOR joint = 0 TO LENGTH(self.B2joints1)

x,y = self.B2joints1[joint]['reference'] GET FORCE

gets the reaction force from joint

totalX = totalX + x

totalY = totalY + y

adds the reaction force to the total

maxForce = maxForce + self.B2joints1[joint]['maxForce']

adds the maxForce of the other material

NEXT

reaction1 = (totalX\*\*2 + totalY\*\*2)\*\*0.5

magnitude of force

totalX = 0

totalY = 0

FOR joint = 0 TO LENGTH(self.B2joints2)

x,y = self.B2joints2[joint]['reference'] GET FORCE

totalX = totalX + x

totalY = totalY + y

maxForce = maxForce + self.B2joints2[joint]['maxForce']

NEXT

reaction2 = (totalX\*\*2 + totalY\*\*2)\*\*0.5

does the same for the other end of the material

maxForce = (maxForce + self.maxForce) / (numOfMaterials + 1)

takes the average of all maxForces

IF reaction1 + reaction2 > maxForce THEN

if the total reaction force is too large it will break

IF reaction1 > reaction2 THEN

if reaction1 is bigger destroy joint1 else destroy joint2

FOR joint = 0 TO LENGTH(self.B2joints1)

WORLD DESTROY(self.B2joints1[joint]['reference'])

destroy joints

NEXT

DELETE self.B2joints1

delete reference to joint

ELSE

FOR joint = 0 TO LENGTH(self.B2joints2)

WORLD DESTROY(self.B2joints2[joint]['reference'])

END IF

DELETE self.B2joints2

END IF

END IF

#### Material.createBody

##### Solid

x1 = self.x1 / 20

x2 = self.x2 / 20

y1 = 30 - self.y1/20

y2 = 30 - self.y2/20

convert pixels to meters

deltax = x2 - x1

deltay = y2 - y1

treat as triangle

length = (deltay^2 + deltax^2)^0.5

angle = tan(deltay/deltax)

get length and angle

pos = (x1+deltax/2,y1+deltay/2)

body position is at the centre of the body

CREATE BODY WITH pos, angle and length

##### Chain

x1 = self.x1 / 20

x2 = self.x2 / 20

y1 = 30 - self.y1/20

y2 = 30 - self.y2/20

deltax = x2 - x1

deltay = y2 - y1

length = (deltay^2 + deltax^2)^0.5

angle = tan(deltay/deltax)

stepx = deltax/10

stepy = deltay/10

distance between each piece in the chain

prevBody = ""

FOR i = 0 TO 10

pos = (x1 + (i+0.5)\*stepx, y1 + (i+0.5)\*stepy)

body = CREATE BODY WITH pos, angle AND length/20

IF i <> 0 THEN

joint CREATE REVOLUTE JOINT between prevBody AND body

self.chainJoints.APPEND(joint)

END IF

self.body.APPEND(body)

prevBody = body

NEXT

## Data Structures

The following data structures are significant to the program and used throughout.

### Stack

There is a single variable that is treated like a stack – materialStack. This variable is used to store the materials the user places when building their bridge. The game will have an undo button with which the user will use to delete the last material they placed. Unfortunately, materialStack isn’t modelled entirely as a stack, if a user decides to delete a joint all related materials will also be deleted, this will delete the material from wherever they are in stack. The stack variable type could have been implemented with object orientation however I did not see it as necessary because there is only one variable treated as a stack and python has built in functions that can be applied to the list, for example materialStack.append(item) will add an material to the list and del materialStack[-1] to remove the last material added.

### Dictionary

Dictionaries are a data type that allow the programmer to access elements in an array using unique identifiers called keys. Python has an in-built dictionary variable type. My uses for dictionaries include storing the bridge, a variable for all buttons in the game and storing joints as the bridge is built or tested. The keys in the dictionary to store the buttons are the names of the button, the button dictionary is then passed to functions to built the required buttons and show them on screen:

buttons = {

'loginOp':Classes.Button('Login',window.get\_rect().centerx,250),

'registerOp':Classes.Button('Register',window.get\_rect().centerx,350),

'quit':Classes.Button('Quit',window.get\_rect().centerx,450),

……

}

buttons[‘loginOP’].create()

### Adjacency List

An adjacency list is a list that is used to store a network. The bridge a user creates will be stored as a network where the joints a vertex and the materials are edges. In python an adjacency list can be treated as a list of dictionaries, each element if the list will be a dictionary that stores what other edge an edge is connected to and the cost of traversal. For the purposes of this program I will alter the structure. An adjacency list showing the structure is in the appendix, this example list will store the information needed to store three joints connected by three pieces of material, following it is an example bridge file from the game.

## SQL Queries

### Python parameterised queries

Parameterised queries are used to prevent SQL injection attacks where a user might enter an sq queriy inside the input form, when their input is queried by the database the query they entered will be performed by the database. In python the method for parameterising queries is using %s notation.

Using %s notation:

Word = “hello”

Name = “John Doe”

Sentencee = “%s world, my name is %s”

Val = (Word,Name)

print(Sentence,Val)

This will output “Hello world, my name is John Doe”.

### Creating the database

CREATE DATABASE BridgeBuilder;

grant usage on \*.\* to Kiran@locahost identified by '!winter@2018';

grant all privileges on BridgeBuilder.\* to Kiran@localhost;

The above query gives user “Kiran” full privileges to define and manipulate data. The database will be called BridgeBuilder and will be accessed using localhost as a test environment. The connection variable will use username “Kiran” and password “!winter@2018”.

### Creating User table

CREATE TABLE User

(

User\_ID INT NOT NULL AUTO\_INCREMENT,

Username VARCHAR(30) NOT NULL,

First\_Name VARCHAR(30) NOT NULL,

Surname VARCHAR(30) NOT NULL,

Email VARCHAR(50) NOT NULL,

Password VARCHAR(128) NOT NULL,

PRIMARY KEY (User\_ID)

);

No variables are null, I am assuming names and usernames will not be longer than 30 characters, the input box in game will prevent the user from entering any more characters. Email addresses can become quite long so its maximum character length is 50. Password requires 128 characters because a hashing algorithm is used that makes the password 128 characters long.

### Creating Bridge table

CREATE TABLE Bridges

(

Bridge\_ID INT NOT NULL AUTO\_INCREMENT,

User\_ID INT NOT NULL,

Bridge\_Name VARCHAR(30) NOT NULL,

Date\_Last\_Edit DATE NOT NULL,

BridgeFile BLOB NOT NULL,

Difficulty VARCHAR(10) NOT NULL,

Land\_Type TINYINT NOT NULL,

PRIMARY KEY (Bridge\_ID),

FOREIGN KEY (User\_ID) REFERENCES User(User\_ID)

);

Date\_Last\_Edit is stored as a date variable, when it is pulled from the database it can be manipulated such that it only returns either the day, month or year. Difficulty can only be “easy”, “normal” or “hard”, it does not need many characters; similarly, Land\_type can only be “1”, “2” or “3” so the tinyint data type is suitable. BridgeFile can become very large and is therefore a Blob variable type.

### findUsername

command = "SELECT \* FROM User WHERE Username = '%s';" %(username)

This query is used to determine whether the username a person has tried to sign up with is taken already.

### findEmail

command = "SELECT \* FROM User WHERE Email = '%s';" %(email)

This query is used to determine whether the email a person has tried to sign up with is taken already.

### addUser

command = ("INSERT INTO User (Username,First\_Name,Surname,Email,Password) VALUES ('%s','%s','%s','%s','%s');"%(username,first,sec,email,password))

This query adds a user’s details to the data base once they have registered

### findUser

command = ("SELECT \* FROM User WHERE Username = '%s' and Password = '%s';"%(username,password))

This query will return all records matching what the user inputs as their login details, either one or none results will be returned as the username is unique, if no results are returned the user entered incorrect details.

### getUser\_ID

command = "SELECT User\_ID FROM User WHERE Username = '%s';" %(username)

This query returns the user ID which will be used throughout the game for other queries.

### findBridge

command = "SELECT \* FROM Bridges WHERE User\_ID = '%s' AND Bridge\_Name = '%s';" %(User\_ID,name)

This query is used when the user enters their bridge name, they must be unique to a user, therefore if this query returns a result the bridge name is taken.

### addBridge

command = ('INSERT INTO Bridges (User\_ID,Bridge\_Name,Date\_Last\_Edit,BridgeFile,Difficulty,Land\_Type) VALUES ("%s","%s",CURDATE(),"%s","%s","%s");'%(info['User\_ID'],name,adjacencyList,info['dif'],info['land']))

Once a user creates a bridge and saves it for the first time this query is used to add the bridge to the database. CURDATE() is a standard mysql function that will add the current date in the database.

### getBridges

command = "SELECT Bridge\_ID,Bridge\_Name,DAYOFMONTH(Date\_Last\_Edit),MONTH(Date\_Last\_Edit),YEAR(Date\_Last\_Edit) FROM Bridges WHERE User\_ID = '%s';" %(User\_ID)

This query returns all bridges belonging to a user when they want to select which bridge they want to delete, edit, or test. The mysql Date variable type can be queried for day, month and year separately so the game can display the date.

### deleteBridges

command = ("DELETE FROM Bridges WHERE Bridge\_ID = '%s'"%(bridgeID))

When a user wishes to delete to delete a bridge they have made, this query will delete the bridge chosen.

### getBridgeFile

command = ("SELECT BridgeFile,Difficulty,Land\_Type FROM Bridges WHERE Bridge\_ID = '%s'"%(bridgeID))

Query is used to get the bridge file for the bridge the user wishes to edit or test.

### updateBridge

command = ('UPDATE Bridges SET BridgeFile = "%s", Date\_Last\_Edit = CURDATE() WHERE Bridge\_ID = "%s";'%(adjacencyList,bridgeID))

If a user is returning to edit a bridge this query is used to update the bridge file.

### getBridgeID

command = ("SELECT Bridge\_ID FROM Bridges WHERE (User\_ID = '%s' AND Bridge\_Name = '%s')"%(ID,name))

Once the user has selected what bridge they would like to test or edit this query will return the bridge ID for identifying the bridge.

# Implementation – important algorithms

## File Structure

This project required 7 separate files:

1. BridgeData.py – this file contains all functions required to connect to the database, retrieving, uploading and deleting data
2. Build.py - contains the algorithms that allow the user to create a bridge
3. Classes.py – contains the OOP classes required throughout the program
4. Graphics.py – contains algorithms required to create the GUI for the user
5. MAIN.py – the initiating file, creates the initial menus of the game
6. Save.py – saves a user’s bridge at request
7. Test.py – creates the bridge in a B2 world and simulates

## Build

### loadBridge

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| bridgeID | Local | Integer | Stores the bridgeID of which bridge is being loaded |
| bridgeFile | Local | Dictionary | Stores a bridge as an adjacency list |
| dif | Local | String | Either “normal”, “easy” or “hard” depending on the difficulty of the loaded bridge |
| Land | Local | Integer | Either 1,2 or 3 depending on the land type the bridge has been built on |
| jointNum | Local | Integer | Number of joints the bridge has |
| materialStack | Local | List | Stores the material objects of the bridge |
| added | Local | Boolean | Used to indicate whether a material has already been added to material stack |
| joint | Local to for loop | String | Stores the key for each element in bridgeFile |
| Joint1 | Local | Integer | Store the index of the joint |
| connectedJoint | Local | Dictionary | Stores the dictionary for a joint that is connected to the current joint in the outer for loop |
| Joint2 | Local | Integer | Store the index of the joint |
| Material | Local to for loop | OOP object | Material object |
| Item | Local | OOP object | Material object |

#### code

def loadBridge(bridgeID):

# define function loadBridge, pass bridgeId as integer

bridgeFile,dif,land = BridgeData.getBridgeFile(bridgeID)

# call for function that retireves and returns relavent infomation about the bridge from the database

bridgeFile = eval(bridgeFile.strip("'"))

# bridgeFile has excess apostrophies applied by the database, use strip function to remove them

# use eval() function to interpret bridgeFile as a dicitonary

jointNum = len(bridgeFile)

# each element in bridgeFile is a joint

materialStack = []

added = False

for joint in bridgeFile:

jointX,jointY = bridgeFile[joint]['location']

jointList.append({'index':int(joint),'point':(jointX\*20,600 - jointY\*20)})

# append dictionary for each joint to jointList, jointX and jointY converted to pixels

for joint in bridgeFile:

joint1 = int(joint)

# define joint1 as integer, assign as the index of current joint in loop

for connectedJoint in bridgeFile[joint]['connectedJoints']:

joint2 = int(connectedJoint['joint'])

# define joint2 as integer, assign as the index of current connected joint in loop

for material in materialStack:

if (material.getJoint1() == joint1 and material.getJoint2() == joint2) or (material.getJoint1() == joint2 and material.getJoint2() == joint1):

added = True

# check if material is already added

if added == False:

if connectedJoint['material'] == 'Steel':

item = Classes.Steel()

elif connectedJoint['material'] == 'Wood':

item = Classes.Wood()

elif connectedJoint['material'] == 'Road':

item = Classes.Road()

elif connectedJoint['material'] == 'Rope':

item = Classes.Rope()

elif connectedJoint['material'] == 'Cable':

item = Classes.Cable()

# if not already in materialStack, new material defined

item.setjoint1(joint1)

item.setjoint2(joint2)

materialStack.append(item)

# add new material to materialStack, assigning joint1 and joint2 attributes

else:

added = False

for material in materialStack:

for joint in jointList:

if int(material.getJoint1()) == int(joint['index']):

jointX,jointY = joint['point']

material.setCo1(jointX,jointY)

elif int(material.getJoint2()) == int(joint['index']):

jointX,jointY = joint['point']

material.setCo2(jointX,jointY)

# set the coordinates of each material, start and end

return [materialStack,jointList,jointNum,dif,land]

### deleteExcessJoint

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| toDelete | Local | List | Stores joints that are to be delete |
| joint | Local to for loop | Dictionary | Stores the location and index of a joint |
| numOf-Connected-Materials | Local | Integer | Number of materials connected to a joint |

#### code

def deleteExcessJoint(jointList,materialStack):

toDelete = []

for joint in jointList:

numOfConnectedMaterials = 0

for material in materialStack:

if joint['index'] == material.getJoint1() or joint['index'] == material.getJoint2():

numOfConnectedMaterials += 1

# checks if joint has any materials about it

if numOfConnectedMaterials == 0:

toDelete.append(joint)

# if joint not connected to any materials joint is added to delete list

for joint in toDelete:

jointList.remove(joint)

# joints deleted from jointList

### Build Loop

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| stickButtons | Local | Dictionary | Stores the button for the build module |
| Info | Local | Dictionary | Stores relevant information about the user and the bridge |
| Window | Local | Pygame object | Stores the reference to the window |
| materialStack | Local | List | Stores the materials the bridge is made of |
| jointList | Local | List | Stores the dictionaries for each joint |
| jointNum | Local | Integer | Stores the number of joints |
| clickStage | Local | String | Stores which button has been clicked/ whether the user has placed the first joint of a piece |
| Click | Local | Boolean | True if mousedown event has been triggered |
| Clock | Local | Pygame clock object | Used to calculate framerate |
| event | Local to for loop | Pygame event object | Stores what event has occurred |
| Material | Local | String | Stores what button has been clicked |
| material | Local to for loop | OOP object | Material object |
| Joint | Local to for loop | Dictionary | Stores index and location of joint |
| Button | Local to for loop | String | Stores each key in stickButtons |
| Item | Local | OOP object | Material object |
| dotX | Local | Integer | Stores the x coordinate of the indicator dot |
| dotY | Local | Integer | Stores the y coordinate of the indicator dot |
| jointPoints | Local | List | stores the locations of all existing joint |
| Cap | Local | Boolean | True if current joint is joint that has been clicked |
| toDelete | Local | List | Stores material objects related to a joint |

#### code

def Main(info,window):

stickButtons = {

'steel':Classes.stickButton(' Steel ',100,550),

'wood':Classes.stickButton(' Wood ',300,550),

'road':Classes.stickButton('Concrete',500,550),

'rope':Classes.stickButton(' Rope ',700,550),

'cable':Classes.stickButton(' Cable ',900,550),

'delete':Classes.stickButton(' Delete ',900,50,(223,12,12)), #red

'quit':Classes.stickButton(' Quit ',100,50,(223,12,12)),

'save':Classes.stickButton(' Save ',300,50,(34,204,0)), #green

'undo':Classes.stickButton(' Undo ',700,50,(223,12,12)),

'test':Classes.stickButton(' Test ',500,50)

}

# creates a dictionary of buttons

if info['loadBridge'] == False:

materialStack = []

jointList = []

jointNum = 0

else:

materialStack,jointList,jointNum,dif,land = loadBridge(info['bridgeID'])

info['dif'] = dif.strip("'")

info['land'] = land

# if new bridge is being created variables are defined otherwise bridge is retrieved from the database

clickStage = "PlaceMaterial1"

click = False

clock = pygame.time.Clock()

while True:

for event in pygame.event.get():

if event.type == QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

click = True

# checks for any events

Graphics.BackDrop(info['dif'],info['land'],window,True)

# creates graphics

Material = buttonOn(stickButtons)

if Material == "delete":

clickStage = "Delete"

elif Material == "quit":

clickStage = "Quit"

click = True

elif Material == "save":

clickStage = "Save"

click = True

elif Material == "undo":

clickStage = "Undo"

click = True

elif Material == "test":

clickStage = "Test"

click = True

# special cases of buttons

for material in materialStack:

material.Draw(window)

for joint in jointList:

Graphics.drawJoint(window,joint['point'])

# draw windows and joints

if clickStage == "PlaceMaterial1":

for button in stickButtons:

stickButtons[button].create(window)

# draw buttons

elif clickStage == "PlaceMaterial2":

item.updateLocation(dotX,dotY)

# update second point of material

item.checkPlacement()

allowConnect(materialStack,item,dotX,dotY)

item.Draw(window)

# if material is being placed buttons are not drawn

if Material != "":

if Material == "delete":

dotX,dotY = Graphics.drawDot(window,info['dif'],info['land'],(233,12,12))

else:

dotX,dotY = Graphics.drawDot(window,info['dif'],info['land'],(255,255,255))

# draw dot when material is being placed

if click:

if clickStage == "PlaceMaterial1":

if Material != "" and stckbtnclk(stickButtons) == "":

# if click has been made to indicate the first postiion of a material

if Material == "steel":

item = Classes.Steel()

elif Material == "wood":

item = Classes.Wood()

elif Material == "road":

item = Classes.Road()

elif Material == "rope":

item = Classes.Rope()

elif Material == "cable":

item = Classes.Cable()

# defines item as selected material

item.setCo1(dotX,dotY)

# sets first coordinate

clickStage = "PlaceMaterial2"

jointPoints = []

for joint in jointList:

jointPoints.append(joint['point'])

# all current joints

if (dotX,dotY) not in jointPoints:

jointNum += 1

jointList.append({'index':jointNum,'point':(dotX,dotY)})

# if joint doesnt already exist add it to jointList

for joint in jointList:

if joint['point'] == (dotX,dotY):

item.setjoint1(joint['index'])

# assings the current materisl's joint1 attribute to the new joint

else:

turnOn(stckbtnclk(stickButtons),stickButtons)

elif clickStage == "PlaceMaterial2":

if item.getError() == False:

jointPoints = []

for joint in jointList:

jointPoints.append(joint['point'])

if (dotX,dotY) not in jointPoints:

jointNum += 1

jointList.append({'index':jointNum,'point':(dotX,dotY)})

for joint in jointList:

if joint['point'] == (dotX,dotY):

item.setjoint2(joint['index'])

materialStack.append(item)

# goes through the same proccess for the second joint but assigns the material's joint2 as the new joint

# appends the new material to materialStack

clickStage = "PlaceMaterial1"

elif clickStage == "Delete":

jointClicked = jointClick(jointList,dotX,dotY)

# which joint is clicked

jointCount = 0

cap = False

toDelete = []

if jointClicked != 0:

for joint in jointList:

if joint['index'] == jointClicked:

cap = True

if cap == False:

jointCount+=1

del jointList[jointCount]

# deletes joint

for material in materialStack:

if material.getJoint1() == jointClicked or material.getJoint2() == jointClicked:

toDelete.append(material)

for material in toDelete:

materialStack.remove(material)

# deletes related materials

deleteExcessJoint(jointList,materialStack)

# delete any joints that dont have any associated materials

clickStage = "PlaceMaterial1"

stickButtons['delete'].turnOff()

elif clickStage == "Quit":

info['build'] = False

return ["sec",info]

# exists module

elif clickStage == "Save":

if len(materialStack) != 0:

Save.Main(window,jointList,materialStack,info)

clickStage = "PlaceMaterial1"

stickButtons['save'].turnOff()

# saves bridge using save module, doesnt save bridge if there are no materials

elif clickStage == "Undo":

if len(materialStack) != 0:

del materialStack[-1]

# delete latest added material

clickStage = "PlaceMaterial1"

stickButtons['undo'].turnOff()

deleteExcessJoint(jointList,materialStack)

# delete joints with no related materials

elif clickStage == "Test":

if len(materialStack) != 0:

Save.Main(window,jointList,materialStack,info)

# saves bridge

info['build'] = False

return ["chooseVehicle",info]

#exits module

clickStage = "PlaceMaterial1"

stickButtons['test'].turnOff()

click = False

clock.tick()

fps = clock.get\_fps()

#print(fps)

pygame.display.update()

## MAIN

### createButtonsLoad

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| Font | Local | Pygame object | Stores the font of text |
| listOfBridgeID | Local | List | Stores the ID of bridges that are on screen |
| listOfBridge-Names | Local | List | Stores the names of bridges that are on screen |
| listOfBridge-Dates | Local | List | Stores the dates of last edit of bridges that are on screen |
| buttonsOn-Screen | Local | Integer | Number of buttons on screen |
| upButton | Local | Boolean | True if up button needs to be displayed on screen |
| downButton | Local | Boolean | True if down button needs to be displayed on screen |
| Results | Local | List | Stores SQL query results from retrieving the user’s bridges |
| Start | Local | Integer | From which element in results list to start displaying bridges |
| End | Local | Integer | From which element in results to stop displaying bridges |
| numOfScreens | Local | Integer | Number of pages required to display all user bridges |
| Bridge | Local to for loop | Integer | Which element in use |
| Height | Local | Integer | Height of warning text box |
| Width | Local | Integer | Width of warining text box |
| X | Local | Integer | X location of warning text box |
| Y | Local | Integer | Y location of warning text box |
| center | Local | Tuple | Location of warning text box |
| Text | Local | String | Text of warning |
| loadMenu | Local | Integer | Which page the user is currently on |
| Buttons | Local | Dictionary | Stores buttons of module |
| Window | Local | Pygame object | Stores window reference |
| User\_ID | Local | Integer | Stores user ID |

#### code

def createButtonsLoad(buttons,window,loadMenu,User\_ID):

# max of 5 bridges per page, buttons will allow scrolling between pages

font = pygame.font.SysFont(None,50)

listOfBridgeID = []

listOfBridgeNames = []

listOfBridgeDates = []

buttonsOnScreen = 0

numOfScreens = 0

upButton = False

downButton = False

results = BridgeData.getBridges(User\_ID)

# all bridges in one list

start = loadMenu\*5

# first bridge on page is in position start

end = start + len(results)%5

# the end is for if the current menu doesnt have 5 bridges

if len(results) != 0:

if len(results)%5 != 0:

numOfScreens = len(results)//5 + 1

if loadMenu == numOfScreens - 1:

# if current page is the final page use start and end as bounds

for bridge in range(start,end):

listOfBridgeID.append(results[bridge][0])

listOfBridgeNames.append(results[bridge][1].replace("'",""))

# list of names will be used to display name

date = ("%s/%s/%s"%(results[bridge][2],results[bridge][3],results[bridge][4]))

# date put into dd/mm/yy format

listOfBridgeDates.append(date)

buttonsOnScreen += 1

else:

for bridge in range(start,start+5):

# if not on final page bounds are simply start and 5 bridges after

listOfBridgeID.append(results[bridge][0])

listOfBridgeNames.append(results[bridge][1].replace("'",""))

date = ("%s/%s/%s"%(results[bridge][2],results[bridge][3],results[bridge][4]))

listOfBridgeDates.append(date)

buttonsOnScreen += 1

else:

numOfScreens = len(results)//5

for bridge in range(start,start+5):

listOfBridgeID.append(results[bridge][0])

listOfBridgeNames.append(results[bridge][1].replace("'",""))

date = ("%s/%s/%s"%(results[bridge][2],results[bridge][3],results[bridge][4]))

listOfBridgeDates.append(date)

buttonsOnScreen += 1

else:

height = 100

width = 600

x = window.get\_rect().centerx

y = window.get\_rect().centery

center = (x,y)

text = Classes.TextBox("You have no bridges",height,width,center)

text.create(window)

# if user has no bridges, message is displayed

if buttonsOnScreen >= 1:

buttons['bridge1'].create(window)

if buttonsOnScreen >= 2:

buttons['bridge2'].create(window)

if buttonsOnScreen >= 3:

buttons['bridge3'].create(window)

if buttonsOnScreen >= 4:

buttons['bridge4'].create(window)

if buttonsOnScreen == 5:

buttons['bridge5'].create(window)

# depending on the number of buttonson screen, buttons are drawn

if loadMenu != 0:

buttons['up'].create(window)

upButton = True

pygame.draw.line(window,White,(500,120),(480,140),10)

pygame.draw.line(window,White,(500,120),(520,140),10)

# if user is not on top page create up button

if loadMenu < numOfScreens-1:

buttons['down'].create(window)

downButton = True

pygame.draw.line(window,White,(500,500),(480,480),10)

pygame.draw.line(window,White,(500,500),(520,480),10)

# if user not on bottom page create down button

buttons['back5'].create(window)

# create back button

for bridgeName in range(0,len(listOfBridgeNames)):

text = font.render(listOfBridgeNames[bridgeName],True,(255,255,255),None)

textRect = text.get\_rect()

textRect.centerx = window.get\_rect().centerx

textRect.centery = 190 + bridgeName\*60

window.blit(text,textRect)

# draw bridge names on the page above the buttons

return [listOfBridgeID,listOfBridgeNames,listOfBridgeDates,buttonsOnScreen,upButton,downButton]

### checkLogin

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| checkList | Local | List | Stores disallowed characters |
| Error | Local | List | Stores error messages |
| Inputboxes | Local | Dictionary | Stores the inputboxes for module |
| User\_ID | Local | Integer | Stores user ID |
| emptyCount | Local | Integer | Number of empty input boxes |
| invalidChar-Count | Local | Integer | Number of input boxes with an invalid character |
| Box | Local to for loop | String | Key in inputboxes |
| Char | Local to for loop | Char | Element in checklist |
| Text | Local | String | Stores error message |
| Hash | Local | Hashlib object | Stores hash function |
| Username | Local | String | Stores inputted username |
| passCap | Local | Integer | Used to store the binary equivalent of the password |
| password | Local | String | Stores hashed password |
| Connect | Local | String | Store status of database query |

#### code

def checkLogin(inputboxes):

checkList = [" ", ";", "=", "'", '"']

# list of characters that are not allowed

error = []

User\_ID = ""

emptyCount = 0

invalidCharCount = 0

for box in inputboxes:

if (box == "Lusername") or (box == "Lpassword"):

for char in checkList:

if inputboxes[box].getCap() == "":

emptyCount += 1

elif char in inputboxes[box].getCap():

invalidCharCount += 1

# checks if any fields are empty or contain disallowed characters

if emptyCount > 0:

text = "All fields must be entered"

error.append(text)

if invalidCharCount > 0:

text = "Incorrect details"

error.append(text)

# append errors to error list that will be displayed in a different function

if error == []:

Hash = hashlib.sha512()

username = inputboxes['Lusername'].getCap()

passCap = bytes(inputboxes['Lpassword'].getCap(), encoding='utf-8')

Hash.update(passCap)

password = Hash.hexdigest()

# creates a hash for the password tp check againt the database

# hashes are one way

connect = BridgeData.findUser(username,password)

# compares password and username to database

if connect == "Not":

error.append("Incorrect details")

elif connect == "Error":

error.append("Error connecting to database")

else:

User\_ID = BridgeData.getUser\_ID(username)

# if details are correct user id is retrieved from the database

if User\_ID == "":

error.append("Error connecting to the database")

return error,User\_ID

### checkRegister

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| CheckList | Local | List | Stores disallowed characters |
| Error | Local | List | Stores error messages |
| emptyCount | Local | Integer | Number of empty inputboxes |
| Box | Local to for loop | String | Key in inputboxes |
| Char | Local to for loop | Char | Element in checklist |
| Text | Local | String | Stores error message |
| username | Local | String | Stores inputted username |
| UsernameState | Local | String | Stores the result of searching for duplicate usernames |
| Email | Local | String | Store inputted email |
| EmailState | Local | String | Stores the result of searching for duplicate emails |
| Hash | Local | Hashlib object | Stores the hash function |
| First | Local | String | Stores the inputted first name |
| Sec | Local | String | Stores the inputted surname name |
| passCap | Local | Integer | Stores the binary equivalent of the password |
| password | Local | String | Stores the hashed password |

#### code

def checkRegister(inputboxes):

checkList = [" ", ";", "=", "'", '"']

error = []

emptyCount = 0

for box in inputboxes:

if (box != "Lusername") and (box != "Lpassword"):

# only two inputboxes not valid to register

for char in checkList:

if (inputboxes[box].getCap() == "") or (char in inputboxes[box].getCap()):

emptyCount += 1

# checks if boxes are empty or contain disallowed characters

if emptyCount > 0:

text = "All fields must be entered and cannot contain ;, =, ' or \""

error.append(text)

if (not("@" in inputboxes['email'].getCap())) and (inputboxes['email'].getCap() != ""):

text = "Please enter a valid email"

error.append(text)

# checks if email is in correct format

if inputboxes['Rpassword'].getCap() != inputboxes['RpasswordC'].getCap():

text = "Passwords do not match"

error.append(text)

# check if passwords match

if error == []:

username = inputboxes['Rusername'].getCap()

UsernameState = BridgeData.findUsername(username)

if UsernameState != "":

if UsernameState == "Taken":

error.append("Username taken")

elif UsernameState == "Error":

error.append("Error connecting to database")

# checks if username is taken

email = inputboxes['email'].getCap()

EmailState = BridgeData.findEmail(email)

if EmailState != "":

if EmailState == "Taken":

error.append("Email taken")

elif EmailState == "Error":

error.append("Error connecting to database")

# checks if email is taken

if error == []:

Hash = hashlib.sha512()

first = inputboxes['first'].getCap()

sec = inputboxes['sec'].getCap()

email = inputboxes['email'].getCap()

username = inputboxes['Rusername'].getCap()

passCap = bytes(inputboxes['Rpassword'].getCap(), encoding='utf-8')

Hash.update(passCap)

password = Hash.hexdigest()

# creates hash for password

connect = BridgeData.addUser(first,sec,email,username,password)

# adds details to database

if connect != "":

error.append("Error connecting to database")

return error

### Main

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| Window | Local | Pygame object | Stores the reference to the window |
| Next | Local | String | Stores the next menu to be displayed when returning to the menu module |
| Info | Local | Dictionary | Stores relevant information about the bridge and the user |

#### code

def main():

# first module called in game

pygame.init()

window = pygame.display.set\_mode((1000,600),0,32)

# initialise pygame and create window

Next = "initial"

# initial is the the name of the first menu screen the user sees

info = {'User\_ID':0,'bridgeID':0,'build':False,'test':False}

# define info dictionary with required initial fields

while True:

if info['build'] == False and info['test'] == False:

info = menuLoop(Next,window,info)

# menus at the start of the game

if info['build']:

Next,info = Build.Main(info,window)

# build module

if info['test']:

Next,info = Test.Main(info,window)

# test module

## Save

### checkSaveName

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| CheckList | Local | List | Stores disallowed characters |
| Error | Local | List | Stores error messages |
| emptyCount | Local | Integer | Number of empty inputboxes |
| Box | Local to for loop | String | Key in inputboxes |
| Char | Local to for loop | Char | Element in checklist |
| Text | Local | String | Stores error message |
| Name | Local | String | Stores inputted name |
| nameState | Local | String | Stores the result of searching for duplicate names |
| filename | Local | String | The name of the file created when saving the bridge |
| File | Local | File object | Used to write to the file |
| openedFile | Local | File object | Used to read the file |
| Connect | Local | String | Store status of database query |
| bridgeID | Local | Integer | Stores the bridge ID |

#### code

def checkSaveName(inputboxes,info,adjacencyList):

checkList = [" ", ";", "=", "'", '"']

# disallowed characters

error = []

emptyCount = 0

bridgeID = ""

for box in inputboxes:

for char in checkList:

if (inputboxes[box].getCap() == "") or (char in inputboxes[box].getCap()):

emptyCount += 1

# checks if boxes are empty or contain invalid character

if emptyCount > 0:

text = "All fields must be entered and cannot contain ;, =, ' or \""

error.append(text)

# append error to display to user in another function

if error == []:

name = inputboxes['name'].getCap()

nameState = BridgeData.findBridge(name,info['User\_ID'])

if nameState == "Taken":

error.append("You have used this name already")

elif nameState == "Error":

error.append("Error connecting to database")

# checks if bridge name has already been used

# tells user if they must change it

if error == []:

fileName = (name+"\_"+str(info['User\_ID']))

with open(fileName,'w') as File:

File.write(str(adjacencyList))

with open(fileName,'r') as File:

openedFile = File.read()

connect = BridgeData.addBridge(name,openedFile,info)

# creates file and saves file to database

os.remove(fileName)

# deletes file

if connect != "":

error.append("Error connecting to database")

else:

bridgeID = BridgeData.getBridgeID(name,info['User\_ID'])

# if successful bridgeID retreived from database

return [error,bridgeID]

### creating adjacencyList / findConnections

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| connectedTo | Local | List | Stores dictionaries of connected joints and with what material the connected is made |
| material | Local to for loop | OOP object | Material object |
| Index | Local | Integer | Joint index |
| adjacencyList | Local | Dictionary | Stores an adjacency list of the bridge |
| Joint | Local | Dictionary | Stores the location and index of a joint |
| connectedJoints | Local | List | Stores what other joints a joint is connected to |

#### code

def findConnections(index,materials):

connectedTo = []

for material in materials:

if index == material.getJoint1():

# if materials first joint is current joint add materials second joint and what material with

connectedTo.append({'joint':material.getJoint2(),'material':material.getMaterial()})

elif index == material.getJoint2():

connectedTo.append({'joint':material.getJoint1(),'material':material.getMaterial()})

return connectedTo

def Main(window,jointList,materialStack,info):

adjacencyList = {}

# define variable as dictionary

for joint in jointList:

# each element in teh dictionary is a joint

index = joint['index']

# the key is the index

x,y = joint['point']

x /= 20

y = 30 - y/20

# coordintes converted to meters

location = (int(x),int(y))

connectedJoints = findConnections(index,materialStack)

adjacencyList[str(index)] = {'location':location, 'connectedJoints':connectedJoints}

# record created, storing infomation on where the joint is,

# what other joints its connected to and with what material

## Test

### createJoints

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| Joint | Local | Dictionary | Stores the location and index of a joints |
| World | Local | PyBox2D world object | Stores the reference to the simulated world |
| groundBody1 | Local | PyBox2D body object | Stores the reference to the first ground body |
| groundBody2 | Local | PyBox2D body object | Stores the reference to the second ground body |
| materialA | Local | OOP object | Material object |
| materialB | Local | OOP object | Material object |
| bodyA | Local | PyBox2D body object | Stores the body of the first body connected to a joint |
| bodyB | Local | PyBox2D body object | Stores the body of the second body connected to a joint |
| Joint | Local | PyBox2D joint object | Stores the reference to the current joint |

#### code

def createJoints(joint,world,groundBody1,groundBody2):

for materialA in joint['materials']:

for materialB in joint['materials']:

# each material connects to every other material about a point in two directions

if materialA != materialB:

if materialA.getMaterial() == "Rope" or materialA.getMaterial()== "Cable":

if joint['index'] == materialA.getJoint1():

bodyA = materialA.getBody()[0]

elif joint['index'] == materialA.getJoint2():

bodyA = materialA.getBody()[-1]

else:

bodyA = materialA.getBody()

# rope and cable bodies consist of a list of 10 bodies

# depending on whether the joint is the material's joint1 or joint2 the PyBox2D joint must

# connect to either the first or last body

if materialB.getMaterial() == "Rope" or materialB.getMaterial()== "Cable":

if joint['index'] == materialB.getJoint1():

bodyB = materialB.getBody()[0]

elif joint['index'] == materialB.getJoint2():

bodyB = materialB.getBody()[-1]

else:

bodyB = materialB.getBody()

Joint = world.CreateRevoluteJoint(bodyA=bodyA,bodyB=bodyB,anchor=joint['point'],collideConnected=False)

# creates revolute joint at the position given by joint['point']

# final parameter allows the materials to collide

if joint['index'] == materialB.getJoint1():

materialB.addB2joints1({"reference":Joint,"maxForce":materialA.getMaxForce()})

elif joint['index'] == materialB.getJoint2():

materialB.addB2joints2({"reference":Joint,"maxForce":materialA.getMaxForce()})

# adds joint to materialB's joint list, used in calculating the reaction force

if joint['onGround1']:

bodyA = groundBody1

elif joint['onGround2']:

bodyA = groundBody2

# checks if the joint is connected to the ground

if joint['onGround1'] or joint['onGround2']:

for material in joint['materials']:

if material.getMaterial() == "Rope" or material.getMaterial()== "Cable":

if joint['index'] == material.getJoint1():

bodyB = material.getBody()[0]

elif joint['index'] == material.getJoint2():

bodyB = material.getBody()[-1]

else:

bodyB = material.getBody()

Joint = world.CreateRevoluteJoint(bodyA=bodyA,bodyB=bodyB,anchor=joint['point'],collideConnected=False)

if joint['index'] == material.getJoint1():

material.addB2joints1({"reference":Joint,"maxForce":material.getMaxForce()})

elif joint['index'] == material.getJoint2():

material.addB2joints2({"reference":Joint,"maxForce":material.getMaxForce()})

# connects each material about that joint to the ground

### loadBridge

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| bridgeID | Local | Integer | Stores the bridge ID of the bridge being retrieved |
| World | Local | PyBox2D world object | Stores the reference to the simulated world |
| materialStack | Local | List | Stores the existing material objects in a bridge |
| jointList | Local | List | Stores the dictionary of each joint |
| jointNum | Local | Integer | Stores the number of joints |
| Dif | Local | String | States the difficulty of the land the bridge is on |
| Land | Local | Integer | States the type of land the material is on |
| Dirt1Y | Local | Integer | Stores the y coordinate of the first ground body |
| Dirt1width | Local | Integer | Stores the width of the first ground body |
| Dirt1height | Local | Integer | Stores the height of the first ground body |
| Dirt2X | Local | Integer | Stores the x coordinate of the second ground body |
| Dirt2Y | Local | Integer | Stores the y coordinate of the second ground body |
| Dirt2width | Local | Integer | Stores the width of the second ground body |
| Dirt2height | Local | Integer | Stores the height of the second ground body |
| Joint | Local to for loop | Dictionary | Stores the location and index of a joint |
| convx | Local | integer | Stores the x variable of a joint so it can be converted to meters |
| convy | Local | integer | Stores the y variable of a joint so it can be converted to meters |
| GB1fix | Local | PyBox2D fixture definition | Stores the body definition of the first ground body |
| groundBody1 | Local | PyBox2D body object | Stores the reference to the body of the first ground body |
| GB2fix | Local | PyBox2D fixture definition | Stores the body definition of the second ground body |
| groundBody2 | Local | PyBox2D body object | Stores the reference to the body of the second ground body |
| wallfix | Local | PyBox2D fixture definition | Stores the body definition of the two boundary walls |
| wall1 | Local | PyBox2D body object | Stores the reference to the body of the first boundary wall |
| wall2 | Local | PyBox2D body object | Stores the reference to the body of the second boundary wall |
| material | Local | OOP object | Material object |
| materialAbout-Joint | Local | List | Stores the materials that connect to the current joint |
| Jointx | Local | Integer | The x coordinate of a joint |
| Jointy | Local | integer | The y coordinate of a joint |

#### code

def loadBridge(bridgeID,world):

materialStack,jointList,jointNum,dif,land = Build.loadBridge(bridgeID)

Dirt1Y,Dirt1width,Dirt1height,Dirt2X,Dirt2Y,Dirt2width,Dirt2height = Graphics.dirtSize(dif,land)

# uses these variables to determine whether a joint is connected to the ground and to create the ground

Dirt1Y = 30 - Dirt1Y/20

Dirt1width = Dirt1width/20

Dirt1height = Dirt1height/20

Dirt2X = Dirt2X/20

Dirt2Y = 30 - Dirt2Y/20

Dirt2width = Dirt2width/20

Dirt2height = Dirt2height/20

# converts to meters

for joint in jointList:

convx,convy = joint['point']

joint['point'] = (convx/20,30-convy/20)

# converts each joint to meters

GB1fix = b2FixtureDef(shape=b2PolygonShape(box=(Dirt1width/2,Dirt1height/2)),friction=0.2,categoryBits=0x0002,maskBits=0x0004)

groundBody1 = world.CreateStaticBody(position=(Dirt1width/2,Dirt1height/2),fixtures = GB1fix)

GB2fix = b2FixtureDef(shape=b2PolygonShape(box=(Dirt2width/2,Dirt2height/2)),friction=0.2,categoryBits=0x0002,maskBits=0x0004)

groundBody2 = world.CreateStaticBody(position=(50-Dirt2width/2,Dirt2height/2),fixtures = GB2fix)

# creates two ground bodies, the left and right island

wallfix = b2FixtureDef(shape=b2PolygonShape(box=(1,15)),friction=0.2,categoryBits=0x0002,maskBits=0x0004)

wall1 = world.CreateStaticBody(position=(-1,15),fixtures = wallfix)

wall2 = world.CreateStaticBody(position=(51,15),fixtures = wallfix)

# creates 2 walls to prevent user's vehicle from falling over the edge

for material in materialStack:

material.createBody(world)

# creates body for each material

for joint in jointList:

materialAboutJoint = []

for material in materialStack:

if joint['index'] == material.getJoint1() or joint['index'] == material.getJoint2():

materialAboutJoint.append(material)

joint['materials'] = materialAboutJoint

# determines what materials connect to current joint

jointx,jointy = joint['point']

if (jointx == Dirt1width and jointy <= Dirt1Y) or (jointx <= Dirt1width and jointy == Dirt1Y):

joint['onGround1'] = True

joint['onGround2'] = False

elif (jointx == Dirt2X and jointy <= Dirt2Y) or (jointx >= Dirt2X and jointy == Dirt2Y):

joint['onGround1'] = False

joint['onGround2'] = True

else:

joint['onGround1'] = False

joint['onGround2'] = False

# determines whether joint is connected to the ground

createJoints(joint,world,groundBody1,groundBody2)

return (materialStack,jointList,dif,land)

### Main

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| World | Local | PyBox2D world object | Stores the reference to the simulated world |
| materialStack | Local | List | Stores the existing material objects in a bridge |
| jointList | Local | List | Stores the dictionary of each joint |
| Dif | Local | String | States the difficulty of the land the bridge is on |
| Land | Local | Integer | States the type of land the material is on |
| Info | Local | Dictionary | Stores relevant information about the user and bridge |
| Clock | Local | Pygame clock object | Used to calculate framerate |
| Timestep | Local | Real | The length of time to simulate with each step |
| Vel\_iters | Local | Integer | Number of iterations to calculate velocity |
| pos\_iters | Local | Integer | Number of iterations to calculate position |
| frameCount | Local | Integer | Number of frames that has passed |
| Font | Local | Pygame object | The font to be used |
| Text1 | Local | Pygame object | Text shown on screen |
| Text1rect | Local | Pygame rect object | Used to set location of text1 |
| Text2 | Local | Pygame object | Text shown on screen |
| Text2rect | Local | Pygame rect object | Used to set location of text2 |
| Buttons | Local | Dictionary | Stores the buttons for the module |
| Vehicle | Local | OOP object | Vehicle object |
| Height | Local | Integer | The height at which the vehicle should be created |
| Forward | Local | Boolean | True if user wants vehicle to move forward |
| Back | Local | Boolean | True if user wants vehicle to move back |
| Click | Local | Boolean | True if user has triggered mousedown event |
| Event | Local to for loop | Pygame event type | Store the triggered event |
| Material | Local | OOP object | Material object |

#### code

def Main(info,window):

world = b2World()

# initialises world

materialStack,jointList,dif,land = loadBridge(info['bridgeID'],world)

# creates bridge

clock = pygame.time.Clock()

# clock varibale used to obtain fps

timeStep = 1.0/25

# time between each simulation step

vel\_iters,pos\_iters = 7,7

# how many times the velocity and position of a body is calculated per step

frameCount = 0

# counts how many frames have passed to determine load time

font = pygame.font.SysFont(None,90)

text1 = font.render('Loading', True,(255,255,255),None)

text1Rect = text1.get\_rect()

text1Rect.centerx = window.get\_rect().centerx

text1Rect.centery = window.get\_rect().centery

text2 = font.render('Loading', True,(0,0,0),None)

text2Rect = text2.get\_rect()

text2Rect.centerx = window.get\_rect().centerx - 5

text2Rect.centery = window.get\_rect().centery - 5

# creates loading text

buttons = {

'quit':Classes.Button(' Quit ',150,50),

'edit':Classes.Button(' Edit ',400,50)

}

# creates buttons

if info['vehicle'] == "car":

vehicle = Classes.car()

elif info['vehicle'] == "bike":

vehicle = Classes.bike()

elif info['vehicle'] == "truck":

vehicle = Classes.truck()

# defines vehicle

if land == 1:

height = 10

elif land == 2:

height = 13

elif land == 3:

height = 8

# determines at what height the vehicles will be created

vehicle.create(world,height)

# creates vehicle

forward,back,click = False,False,False

# defines variables for vehicle movement

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

click = True

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_d:

if back == False:

forward = True

# d to move forwards

elif event.key == pygame.K\_a:

if forward == False:

back = True

# a to move back

elif event.type == pygame.KEYUP:

if event.key == pygame.K\_d:

forward = False

elif event.key == pygame.K\_a:

back = False

if frameCount%5 == 0 and frameCount > 100:

for material in materialStack:

material.ifBreak(world,timeStep)

# if not loading determine whether material will break

# functions within ifBreak are costly in processing power only done once every 5 steps

Graphics.BackDrop(dif,land,window)

# creates graphics

for material in materialStack:

material.testDraw(window)

# draws materials

vehicle.Draw(window)

# draws vehicle

if frameCount > 100:

if forward:

vehicle.forward()

elif back:

vehicle.back()

elif (forward == False and back != True) or (back == False and forward != True):

vehicle.stop()

# if not loading allow user to control vehicle

else:

window.blit(text2,text2Rect)

window.blit(text1,text1Rect)

frameCount += 1

# if loading outing "Loading text"

buttons['quit'].create(window)

buttons['edit'].create(window)

# creates buttons on screen

if click:

if buttons['quit'].ifClick():

info['test'] = False

return ["sec",info]

# if quit is clicked take user to play menu

elif buttons['edit'].ifClick():

info['test'] = False

info['loadBridge'] = True

info['build'] = True

return ["build",info]

# edit is clicked take user to build

world.Step(timeStep,vel\_iters,pos\_iters)

# simulate world step

world.ClearForces()

# clears applied forces

#clock.tick()

#print(clock.get\_fps())

# outputs fps

pygame.display.update()

pygame.time.wait(5)

# update display

## Classes

### Material.ifBreak

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| World | Local | PyBox2D world object | Stores the reference to the simulated world |
| timestep | Local | Real | The length of time to simulate with each step |
| totalX | Local | Integer | The total horizontal component of force |
| totally | Local | Integer | The total vertical component of force |
| numOfJoints | Local | Integer | The number of Pybox2D joints connected to a material |
| maxForce | Local | Integer | The average of the maxForce of all materials about a joint |
| Reaction1 | Local | Integer | The magnitude of the force on joint1 |
| Reaction2 | Local | Integer | The magnitude of the force on joint2 |
| toDelete | Local | Integer | The joint to delete in the chain |
| Joint | Local to for loop | Integer | The number of the element in chainJoints |

#### Solid

def ifBreak(self,world,timeStep):

totalX = 0

totalY = 0

numOfJoints = len(self.B2joints1) + len(self.B2joints2)

maxForce = 0

for joint in self.B2joints1:

x,y = joint['reference'].GetReactionForce(1/timeStep)

# gets the reaction force from joint

totalX += x

totalY += y

# adds the reaction force to the total

maxForce += joint['maxForce']

# adds the maxForce of the other material

reaction1 = (totalX\*\*2 + totalY\*\*2)\*\*0.5

# magnitude of force

totalX = 0

totalY = 0

for joint in self.B2joints2:

x,y = joint['reference'].GetReactionForce(1/timeStep)

totalX += x

totalY += y

maxForce += joint['maxForce']

reaction2 = (totalX\*\*2 + totalY\*\*2)\*\*0.5

# does the same for the other end of the material

maxForce = (maxForce + self.maxForce) / (numOfJoints + 1)

# takes the average of all maxForces

if reaction1 + reaction2 > maxForce:

# if the total reaction force is too large it will break

if reaction1 > reaction2:

# if reaction1 is bigger destroy joint1 else destroy joint2

for joint in self.B2joints1:

world.DestroyJoint(joint['reference'])

# destroy joints

del self.B2joints1[:]

# delete reference to joint

else:

for joint in self.B2joints2:

world.DestroyJoint(joint['reference'])

del self.B2joints2[:]

#### Chain

def ifBreak(self,world,timeStep):

Material.ifBreak(self,world,timeStep)

# use polymorphism and call for the supers function

toDelete = -1

for joint in range(0,len(self.chainJoints)):

forceX,forceY = self.chainJoints[joint].GetReactionForce(1/timeStep)

force = (forceX\*\*2 + forceY\*\*2)\*\*0.5

if force > self.maxForce:

toDelete = joint

if toDelete != -1:

world.DestroyJoint(self.chainJoints[toDelete])

del self.chainJoints[toDelete]

# checks if any joint inside the chain is experiencing a large force

# breaking it if the force is too large

### Material.createBody

#### data dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Scope** | **Variable type** | **Used for** |
| World | Local | PyBox2D world object | Stores the reference to the simulated world |
| x1 | Local | Integer | x location of joint1 of material |
| x2 | Local | Integer | x location of joint2 of material |
| y1 | Local | Integer | y location of joint1 of material |
| y2 | Local | Integer | y location of joint2 of material |
| deltax | Local | Integer | Difference in x1 and x2 of material |
| deltay | Local | Integer | Difference in y1 and y2 of material |
| length | Local | Real | Length of material |
| Angle | Local | Real | Angle of material against positive x direction in radians |
| pos | Local | Tuple | Location of entre of material |
| Fix | Local | PyBox2D fixture definition | Stores the fixture of the material |
| stepx | Local | Real | The chain in x location for joints and bodies in chain |
| stepy | Local | Real | The chain in y location for joints and bodies in chain |
| prevBody | Local | PyBox2D body object | The previous body in the chain |
| body | Local to for loop | PyBox2D body object | The current body in the chain |
| posX | Local | Integer | The x position of the current body |
| posY | Local | Integer | The y position of the current body |

#### Solid

def createBody(self,world):

x1 = self.x1 / 20

x2 = self.x2 / 20

y1 = 30 - self.y1/20

y2 = 30 - self.y2/20

# convert pixels to meters

deltax = x2 - x1

deltay = y2 - y1

# treat as triangle

length = (deltay\*\*2 + deltax\*\*2)\*\*0.5

angle = math.atan2(deltay,deltax)

# get length and angle

pos = (x1+deltax/2,y1+deltay/2)

# body position is at the centre of the body

fix = b2FixtureDef(shape=b2PolygonShape(box=(length/2,0.2)),density=self.density,friction=0.2,categoryBits=0x0002,maskBits=0x0004)

fix.filter.groupIndex = self.groupIndex

# set collision filter so it only collides with certain materials

self.body = world.CreateDynamicBody(position=pos,fixtures=fix,angle=angle)

# create body

#### Chain

def createBody(self,world):

x1 = self.x1 / 20

x2 = self.x2 / 20

y1 = 30 - self.y1/20

y2 = 30 - self.y2/20

# convert to meters

deltax = x2 - x1

deltay = y2 - y1

# treat as triangle

length = (deltay\*\*2 + deltax\*\*2)\*\*0.5

angle = math.atan2(deltay,deltax)

# get length and angle of material

stepx = deltax/10

stepy = deltay/10

# distance between each piece in the chain

fix = b2FixtureDef(shape=b2PolygonShape(box=(length/20,0.35)),density=self.density,friction=0.2,categoryBits=0x0002,maskBits=0x0004)

fix.filter.groupIndex = self.groupIndex

# all pieces in the chain have the same fixture

prevBody = ""

for i in range(0,10):

pos = (x1 + (i+0.5)\*stepx, y1 + (i+0.5)\*stepy)

body = world.CreateDynamicBody(position=pos,fixtures = fix,angle = angle)

# create body

if i != 0:

joint = world.CreateRevoluteJoint(bodyA=prevBody,bodyB=body,anchor=(x1 + i\*stepx, y1 + i\*stepy))

self.chainJoints.append(joint)

# create joint

self.body.append(body)

prevBody = body

# Testing

Test Type Key:

* Err – erroneous: inputs that should produce an error and be handled
* Ext – extreme: extremely large inputs that should cause a run time error
* Bnd – boundary: testing the limits of conditions
* Norm – normal: inputs that should be accepted
* Exc – exceptional: [space], null, 0 etc

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Section of program** | **(Test type) What is being tested** | **Action/input** | **Expectation** | **S/F** | **Notes** | **Appendix number** |
| Starting program | (Norm) If the program turns on | Inputting into console to run the program | Programs starts | S | White box tested | 1 |
| Closing program | (Norm) Terminating program using close button | Clicking the close button (red cross) | Program terminates | S | White box tested | 2 |
| Initial menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 3 |
| Initial menu | (Norm) Graphics functionality of buttons | Hover above the ‘Login’ button | Button should become highlighted | S | White box tested | 4 |
| Initial menu | (Norm) Graphics functionality of buttons | Hover above the ‘Register’ button | Button should become highlighted | S | White box tested | 5 |
| Initial menu | (Norm) Graphics functionality of buttons | Hover above the ‘Quit’ button | Button should become highlighted | S | White box tested | 6 |
| Initial menu | (Norm) Interactive functionality of buttons | Clicking the ‘Login’ button | The program will proceed to the Login menu | S | White box tested | 7 |
| Initial menu | (Norm) Interactive functionality of buttons | Clicking the ‘Register’ button | The program will proceed to the Register menu | S | White box tested | 8 |
| Initial menu | (Norm) Interactive functionality of buttons | Clicking the ‘Quit’ button | The program will terminate | S | White box tested | 9 |
| Register menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 10 |
| Register menu | (Norm)  Functionality of buttons | Hover above the “Back” button | The button will change colour | S | White box tested | 11 |
| Register menu | (Norm) functionality of buttons | Click the “Back” button | The game should return to the initial menu | S | White box tested | 12 |
| Register menu | (Norm) functionality of input boxes | Click any input box | The box should become a different colour | S | White box tested, the input boxes are built as objects they will all react the same way | 13 |
| Register menu | (Norm) functionality of input boxes | Click anywhere else | The box will be deselected | S | White box tested, the input boxes are built as objects they will all react the same way | 14 |
| Register menu | (Norm) functionality of input boxes | Any character input into text box | The character will be shown on screen and remains there if user activity changes | S | White box tested, the input boxes are built as objects they will all react the same way | 15 |
| Register menu | (Norm) functionality of buttons | Any character input into password box | Xs will be shown, hiding the password | S | White box tested, the input boxes are built as objects they will all react the same way | 16 |
| Register menu | (Bnd) functionality of buttons | Enter large quantities of characters into each box | All boxes will cap at 20 characters except for the email box which is capped at 50, if required characters will go beyond the box | S | White box tested, once the limit is reached the user is no loner able to type characters into the box | 17 |
| Register menu | (Err) disallowed inputs into register | Complete null input, click register button | Appropriate error message will be displayed | S | White box tested | 18 |
| Register menu | (Norm) functionality of buttons | Click the ok button after an error is displayed | Error message disappears | S | White box tested | 19 |
| Register menu | (Err) disallowed inputs into register | Partial null input, click register button | Appropriate error message will be displayed | S | White box tested | 20 |
| Register menu | (Err) disallowed inputs into register | Barely null input, click register button | Appropriate error message will be displayed | S | White box tested | 21 |
| Register menu | (Err) disallowed inputs into register | Disallowed characters inputted into register | Appropriate error message will be displayed | S | White box tested | 22 |
| Register menu | (Err) disallowed inputs into register | Input not in the form of an email entered in email box | Appropriate error message will be displayed | S | White box tested | 23 |
| Register menu | (Err) disallowed inputs into register | Inputs into password and retype boxes are different | Appropriate error message will be displayed | S | White box tested | 24 |
| Register menu | (Err) disallowed inputs into register | All errors inputted simultaneously | Appropriate error message will be displayed | S | White box tested | 25 |
| Register menu | (norm) creating account | All inputs are valid | User taken to login page | F | White box tested, fail safe triggered | 26 |
| Regsiter menu | (norm) finding solution to test number 26 | All inputs are valid | One of three potential errors will be displayed | N/A | There are three possible places this error can be produced, I have change the error message to indicate which one is triggering the error. The error occurs when the search for whether the username is already taken is carried out. There was a logic error in the sql query, “User” was used rather than “Username”. | 27 |
| Register menu | (norm) creating account retest | All inputs are valid | User taken to login page | F | Same fail safe triggered, this time the error comes from writing the data to the database. The error occurred due to the size of the password string, the database allowed for a password 64 characters lng but the hash used made the password 128 characters long. | 28 |
| Register menu | (norm) creating an account retest | All inputs are valid | User taken to login page | S | Black box test, with the altered database the error is eliminated | 29 |
| Login menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 30 |
| Login menu | (norm) functionality of input boxes | Click on the input box | The box should change colour | S | White box tested | 31 |
| Login menu | (norm) functionality of input boxes | Random click | The box will revert back to its original colour | S | White box tested | 32 |
| Login menu | (norm) functionality of buttons | Hovering above the “back” button | It will change colour | S | White box tested | 33 |
| Login menu | (norm) functionality of buttons | Clicking on the “back” button | The game will take the user back to the initial menu | S | White box tested | 34 |
| Login menu | (norm) functionality of input boxes | Any character can be inputted | “Username” box will show the characters, “Password” box will show Xs for each character entered | S | White box tested | 35 |
| Login menu | (bnd) functionality of input boxes | Large amounts of characters entered | The user will no longer be able to enter any characters once they type 20 characters, if required the characters will go beyond the box | S | White box tested | 36 |
| Login menu | (err) logging in | Inputting incorrect details, clicking “login” button | Appropriate error message will be shown | S | White box tested | 37 |
| login menu | (norm) logging in | Inputting correct details, clicking “login” button | The user will be taken to the main menu | S | White box tested | 38 |
| Login menu | (err) logging in | Inputting a correct username but incorrect password | Appropriate error message will be displayed | S | White box tested | 39 |
| Login menu | (err) logging in | Inputting a correct password but incorrect usrename | Appropriate error message will be displayed | S | White box tested | 40 |
| Main menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 41 |
| Main Menu | (norm) functionality of buttons | Hover above the “logout” button | The button should change colour | S | White box tested | 42 |
| Main menu | (norm) functionality of buttons | Click the “logout” button | The user will be taken back to the initial menu | S | White box tested | 43 |
| Main menu | (norm) functionality of buttons | Hover above the “Instructions” button | The button will change colour | S | White box tested | 44 |
| Main menu | (norm) functionality of buttons | Click the “instruction” button | The user will be displayed with the instructions on what to do with the game | S | White box tested, the instructions will be implemented at a later date | 45 |
| Instructions screen | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 46 |
| Instructions screen | (norm) functionality of buttons | Hover above the “Back” button | The button will change colour | S | White box tested | 47 |
| Instructions screen | (norm) functionality of buttons | Click the “back” button | The user will be taken back to the main menu | S | White box tested | 48 |
| Main menu | (norm) functionality of buttons | Hovering above the “play” button | The button will change colour | S | White box tested | 49 |
| Main menu | (norm) functionality of buttons | Clicking the “play” button | The user will be taken to a further menu | S | White box tested | 50 |
| Play menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 51 |
| Play menu | (norm) functionality of buttons | Hovering above the “back” button | The button should change colour | S | White box tested | 52 |
| Play menu | (norm) functionality of buttons | Clicking the “back” button | The user will be taken back to the main menu | S | White box tested | 53 |
| Play menu | (norm) functionality of buttons | Hovering above the “Load Bridge” button | The button will change colour | S | White box tested | 54 |
| Play menu | (norm) functionality of buttons | Hovering above the “New Bridge” button | The button will change colour | S | White box tested | 55 |
| Play menu | (norm) functionality of buttons | Clicking the “New Bridge” button | The user will be given the choice of what landscape they want to build on | S | White box tested | 56 |
| Land menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 57 |
| Land menu | (norm) functionality of buttons | Hovering above the “Back” button | The button will change colour | S | White box tested | 58 |
| Land menu | (norm) functionality of buttons | Hovering above the “High to Low” button | The button will change colour | S | White box tested | 59 |
| Land menu | (norm) functionality of buttons | Hovering above the “Flat” button | The button will change colour | S | White box tested | 60 |
| Land menu | (norm) functionality of buttons | Hovering above the “Low to High” button | The button will change colour | S | White box tested | 61 |
| Land menu | (norm) functionality of buttons | Clicking on any button other than the “back” button | Either button will take the user to the next menu where they can choose the difficulty with specific graphics | S | White box tested | 62 |
| Difficulty menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 63 |
| Difficulty menu | (norm) functionality of buttons | Hovering above the “back” button | The button will change colour | S | White box tested | 64 |
| Difficulty menu | (norm) functionality of buttons | Clicking the “back” button | The user will be taken back to the land menu | S | White box tested | 65 |
| Difficulty menu | (norm) functionality of buttons | Hovering above the “Easy” button | The button will change colour | S | White box tested | 66 |
| Difficulty menu | (norm) functionality of buttons | Hovering above the “Normal” button | The button will change colour | S | White box tested | 67 |
| Difficulty menu | (norm) functionality of buttons | Hovering above the “Hard” button | The button will change colour | S | White box tested | 68 |
| Difficulty menu | (norm) functionality od buttons/ graphics test | Clicking on any difficulty and land/ if graphics are created correctly | The user will be taken to the build screen with chosen options/buttons will be in the correct place, land lines up correctly with grid | F | White box tested, land does not line up with grid | 69 |
| Difficulty menu | (norm) functionality od buttons/ graphics test  Retest | Clicking on any difficulty and land/ if graphics are created correctly | The user will be taken to the build screen with chosen options/buttons will be in the correct place, land lines up correctly with grid | S | White box tested, to fix this I divided the height of the dirt by 20 (number of pixels between grid lines) rounded it up ((0,0) at top left) and multiplied it by 20 again | 70 |
| Build | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 71 |
| Build | (norm) functionality of buttons | Clicking the material buttons | The buttons will turn on | S | White box tested | 72 |
| Build | (norm) functionality of buttons | Clicking the quit button | The user will be taken | S | White box tested | 73 |
| Build | (norm)  Position dot functionality | Dragging the mouse across the screen (with a material selected) | The dot will follow the mouse staying on the cross sections of the grid | S | White box tested | 74 |
| Build | (norm)  Position dot functionality | Dragging the mouse above dirt (with a material selected) | The dot will not follow the mouse above dirt and stay on the closest cross section to the mouse | S | White box tested | 75 |
| Build | (exc)  Position dot functionality | Move the mouse of the screen | The dot will remain at the edge of the screen | S | White box tested | 76 |
| Build | (norm) functionality of materials | A click on screen after a material is selected | The material will appear on screen and follow the position dot, the buttons will not be on display | S | White box tested | 77 |
| Build | (norm) functionality of materials | After the first point of a material is placed, the mouse is dragged away | Once far enough the material will turn red as it has reached its length limit | S | White box tested | 78 |
| Build | (norm) functionality of materials | Another click after the first point of a material is chosen | The material will be placed, and other materials can be placed | S | White box tested | 79 |
| Build | (norm) functionality of materials | Using other materials | All materials work this way with the correct colours and lengths | S | White box tested | 80 |
| Build | (norm) functionality of button | Clicking on the undo button | The last added material will be deleted | S | White box tested | 81 |
| Build | (norm) functionality of button/position dot | Clicking on the delete button and deleting a joint | The buttons will be hidden, and the position dot will become red, if a joint is clicked on the joint and any related materials will be deleted | S | White box tested | 82 |
| Build | (exc) functionality of position dot | Clicking on anything but a joint after the delete button is selected | The buttons will be displayed, and nothing is deleted | S | White box tested | 83 |
| Build | (norm) functionality of buttons | Clicking on the save button | An input box will be displayed | S | White box tested | 84 |
| Build | (norm) functionality of buttons | Hovering above the cancel button | The button will change colour | S | White box tested | 85 |
| Build | (norm) functionality of buttons | Hovering above the save button | The button will change colour | S | White box tested | 86 |
| Build | (norm) functionality of input boxes | Clicking on the input box | The box will change colour | S | White box tested | 87 |
| Build | (norm) functionality of input boxes | Typing in the input box | The characters will appear in the box | S | White box tested | 88 |
| Build | (bound) functionality of input boxes | Typing in the input box to its limit | The user will be unable to continue typing | S | White box tested | 89 |
| Build | (norm) functionality of button | Clicking the cancel button | The input box will disappear, and nothing will be saved | S | White box tested | 90 |
| Build | (norm) functionality of button | Clicking the save button | The input box will disappear, and the bridge will be saved | S | White box tested | 91 |
| Play Menu | (norm) functionality of button | Clicking on the “Load Bridge” button | The user will be displayed with their bridges | S | White box tested | 92 |
| Load menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 93 |
| Load Menu | (norm) functionality of button | Hover above the “Back” button | The button will change colour | S | White box tested | 94 |
| Load Menu | (norm) functionality of button | Hover above the bridge button | The button will change colour | S | White box tested | 95 |
| Load Menu | (norm) functionality of button | Clicking on the “Back” button | The user will be taken to the previous menu | S | White box tested | 96 |
| Load Menu | (norm) functionality of button | Clicking on the bridge button | The user will be given options on what to do with the bridge | S | White box tested | 97 |
| Bridge option menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 98 |
| Bridge option menu | (norm) functionality of button | Hover above the “Edit” button | The button will change colour | S | White box tested | 99 |
| Bridge option menu | (norm) functionality of button | Hover above the “Test” button | The button will change colour | S | White box tested | 100 |
| Bridge option menu | (norm) functionality of button | Hover above the “Delete” button | The button will change colour | S | White box tested | 101 |
| Bridge option menu | (norm) functionality of button | Hover above the “Back” button | The button will change colour | S | White box tested | 102 |
| Bridge option menu | (norm) functionality of button | Click on the “Back” button | The user will be taken to the previous menu | S | White box tested | 103 |
| Bridge option menu | (norm) functionality of button | Click on the “Edit” button | The user will be taken to the build menu | S | White box tested | 104 |
| Bridge option menu | (norm) functionality of button | Click on the “Delete” button | A warning message will show | S | White box tested | 105 |
| Bridge option menu | (norm) functionality of button | Hover above the “Cancel” button | The button will change colour | S | White box tested | 106 |
| Bridge option menu | (norm) functionality of button | Hover above the “Yes” button | The button will change colour | S | White box tested | 107 |
| Bridge option menu | (norm) functionality of button | Click on the “Cancel” button | The warning message will disappear, and nothing is deleted | S | White box tested | 108 |
| Bridge option menu | (norm) functionality of button | Click on the “Yes” button | The user is taken to the previous menu, a message will be displayed because there are no bridges left | S | White box tested | 109 |
| Load menu | (bound)  How the program handles many bridges | Creating more bridges | There will be a button for each bridge and beyond 5 bridges a button to scroll down | S | White box tested | 110 |
| Load menu | (norm) functionality of button | Hover above the down button | The button will change colour | S | White box tested | 111 |
| Load menu | (norm) functionality of button | Click on the down button | The menu will “scroll” down | S | White box tested | 112 |
| Load menu | (norm) functionality of button | Hover above the up button | The button will change colour | S | White box tested | 113 |
| Load menu | (norm) functionality of button | Click on the up button | The menu will “scroll” up | S | White box tested | 114 |
| Load menu | (exc) the program reacts correctly to a scroll button not being required | Delete bridge 6 | The scroll menu will go to the top | S | White box tested | 115 |
| Build | (norm) saving an already created bridge | Saving a bridge that already has a name | Nothing will be shown to the user it will only update the database | S | White box tested | 116 |
| Build | (norm) saving a new bridge | Trying to save the name as one that is already taken | An error message will show | F | White box tested, error shown in screenshot | 117 |
| Build | (norm) saving a bridge retest | Trying to save the name as one that is already taken | An error message will show | S | Black box tested, the bridgeID variable was assigned inside an if statement and was therefore missed and caused an error when it was called for | 118 |
| Build | (norm) functionality of button | Hover above the “OK” button | The button will change colour | S | White box tested | 119 |
| Build | (norm) functionality of button | Click on the “OK” button | The error message will disappear | F | White box tested,  The error message does not disappear | 120 |
| Build | (norm) functionality of button retest | Click on the “OK” button | The error message and input box will disappear | F | White box tested, the input box and error message are printed on top of the background which is only changed inside of another function, because of how I have created the program it is not possible to print over the message unless the save function is exited | 121 |
| Build | (norm) functionality of buttons | Clicking on the save button with no materials placed | Nothing will happen, nothing is saved | S | White box tested | 122 |
| Build | (norm) functionality of buttons | Clicking on the test button with no materials placed | Nothing will happen, nothing is saved, player is not taken to test menu | S | White box tested | 123 |
| Build | (norm) functionality of buttons | Clicking on the test button with materials placed | The user will be asked to save if the bridge is not saved already, after saved the user will be taken to the vehicle menu | S | White box tested | 124 |
| Load menu | (norm) functionality of buttons | Click on the test button | The user will be taken to the vehicle menu | S | White box tested | 125 |
| Vehicle menu | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 126 |
| Vehicle menu | (norm) functionality of buttons | Hover above the “Bike” button | The button should change colour | S | White box tested | 127 |
| Vehicle menu | (norm) functionality of buttons | Hover above the “Car” button | The button should change colour | S | White box tested | 128 |
| Vehicle menu | (norm) functionality of buttons | Hover above the “Truck” button | The button should change colour | S | White box tested | 129 |
| Vehicle menu | (norm) functionality of buttons | Hover above the “Back” button | The button should change colour | S | White box tested | 130 |
| Vehicle menu | (norm) functionality of buttons | Clicking on the “Back” button | The user will be taken back to the load menu | S | White box tested | 131 |
| Vehicle menu | (norm) functionality of buttons | Clicking on the “Bike” button | The user will be taken to the testing part of the program, their bridge, chosen vehicle and loading sign will show | S | White box tested | 132 |
| Vehicle menu | (norm) functionality of buttons | Clicking on the “Car” button | The user will be taken to the testing part of the program, their bridge, chosen vehicle and loading sign will show | S | White box tested | 133 |
| Vehicle menu | (norm) functionality of buttons | Clicking on the “Truck” button | The user will be taken to the testing part of the program, their bridge, chosen vehicle and loading sign will show | S | White box tested | 134 |
| Testing | (norm) loading | Doing nothing | “loading” will disappear | S | White box tested | 135 |
| Testing | (Exc) Program reacts correctly to null click | A random click on the page where there is no button whilst loading | Nothing should happen | S | White box tested | 136 |
| Testing | (Exc) Program reacts correctly to null click | A random click on the page where there is no button | Nothing should happen | S | White box tested | 137 |
| Testing | (norm) functionality of button | Hovering above the “Quit” button | The button should change colour | S | White box tested | 138 |
| Testing | (norm) functionality of button | Hovering above the “Edit” button | The button should change colour | S | White box tested | 139 |
| Testing | (norm) functionality of button | Clicking the “Back” button | The user will be taken to the play menu | S | White box tested | 140 |
| Testing | (norm) functionality of button | Clicking the “Edit” button | The user will be taken to the build menu with their bridge | S | White box tested | 141 |
| Testing | (norm) functionality of vehicle and bridge | Pressing D key to move right | The car will pass over the bridge | S | White box tested | 142 |
| Testing | (norm) functionality of vehicle and bridge | pressing A key to move right | The car will pass over the bridge | S | White box tested | 143 |
| Testing | (bnd) functionality of vehicle | Keeping A down | The car will hit a boundary | S | White box tested | 144 |
| Testing | (bnd) functionality of vehicle | Keeping D down | The car will hit a boundary | S | White box tested | 145 |
| Testing | (norm) functionality of bridge | Driving a vehicle across a weak bridge | The bridge will collapse | F | White box tested, game crashes | 146 |
| Testing | (norm) functionality of bridge retest | Driving a vehicle across a weak bridge | The bridge will collapse | S | Black box tested, the problem arose in destroying the joint, the reference (variable) to the joint must also be deleted, the joint was stored in a list so the list must be deleted | 147 |
| Instructions | (norm) graphics |  | User can see instructions | S | White box tested | 148 |
| Register | (norm) prevention of SQL injection | Input “); DROP TABLE Bridges;” | Input is not aloud error message displayed | S | White box tested | 149 |

# Evaluation

## SMART criteria completion

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Sub Task** | **Completed** | **Tests** |
| 1.When the program is first turned on the user will be presented with a menu | a.The first option is to log on; the user will be asked to enter their username and password, if any details are incorrect the user will be alerted, and they will have to re-enter their details | Yes | 7, 37-38 |
|  | b.The second option is to register; the user will be required to enter their name, email and desired username and password, the password will have a second box to check the password is correct. If the username or email is already taken or the passwords do not match the user will have to re-enter their details or alter them appropriately. | Yes | 8, 25,29 |
|  | c.The third option is to quit the game, this will simply leave stop the program from running. | Yes | 9 |
| 2.Once logged on the user will be presented with a further 3 options | a.The first option named “Start” will take the user to a further menu . | Yes. Alternative name given to button | 50 |
|  | b.The second option will show the user what the objective of the game is and how to play the game | Yes | 148 |
|  | c.The third option will log the user out taking them back to the first menu | Yes | 43 |
| 3.After the “Start” option is selected the user will be displayed with a further menu | a.The first option will be to create a new bridge: the user will be given a series of options to select the difficulty and shape of the land they will be building on. | Yes, the series of options are displayed as a further 2 menus | 56-69 |
|  | b.The second option will give users access to their previously built bridges so they can either edit them or test them. | Yes | 92-115 |
|  | c.The third option will simply return the user to the previous menu. | Yes | 53 |
| 4.The user will be able to build on the landscape they have chosen. | a.The user will be able to select from a small variety of materials to build from, they will be able to select where they place their material on a grid | Yes | 77-81 |
|  | b.The user will be able to undo the last build they made and delete any previous materials placed | Yes, the user can delete previously placed joints | 82-83 |
|  | c.The user will also be able to save their bridge (giving it a name if it already doesn’t already have one) and click a test button that takes them directly to the testing screen with their bridge | Yes | 84-91,124 |
| 5.The user will be able to test their bridge | a.The user can choose from a small selection of vehicles to test their bridge with | Yes, the vehicles mainly vary in weight | 126-134 |
|  | b.The vehicle will be user controlled | Yes | 142-143 |
|  | c.If the bridge breaks the user can return to the build menu to edit their bridge | Yes | 141,147 |

## User Feed Back

### Student 1

It’s a good game. Everything is easy to access. If bridge fails there could be better feed back to the user, or maybe give a score out of 10 rating the bridges stability

### Student 2

I would appreciate more keyboard short cuts such as tab to enter into the next input box. I would approve if the frame could be raised. I think it would be a good idea that the user could place materials at places other than joints. It would be fun if there were more animations.

### Student 3

Easy to understand layout; clear fonts used. I was able to enter the invalid email “@111” during the register stage which should not be possible. I enjoyed the gameplay, but it would help if the keys to control the speed of the vehicle could be displayed to the user during the bridge testing stage. Finally, whilst the implementation on the whole was good, employing methods to increase the framerate should be attempted if this project was to be continued beyond its current state.

### Student 4

The idea of this game is fantastic with different levels and difficulties there is variety of gameplay. The core mechanics of the game are impeccable including the ability to use different materials to form your bridge. As constructive criticism I would suggest the game designer improves the user interface, by placing the instructions of how things are done in a more appropriate place.

### Student 5

The game is easy to play and features a range of difficult which help enhance individual player experience. The application of physics on the forces acting on the bridge brings the game into real life, providing it with the reality it needs to enthuse the gamer, as the objective is made harder.

### Student 6

Easy to get the hang of and would be fun to try out different bridges good game!

## Future Development

### GUI

The GUI is not the focus for this project, however it can be developed further in the future. Pygame was used a simple and easy to use library for a simple UI. Some simple animations for water can be implemented in a sinusoidal wave rather than several circles connected together. Due to not being able to rotate rect objects in Pygame any rectangle drawn at an angle is made from lines: the material are just thick lines, the car’s chassis consists of 4 lines e.g. As for keyboard shortcuts, they could be implemented quite easily, an extra function added such that when the tab button is entered the current input box is turned off and the next one is turned on; or so that when the enter button is pressed the login/ register button is pressed

### Online integration

Currently there is no use for the user email, if this game were to be implemented online, the users’ email is good way to keep users unique and features such as “forgotten password” can be added. Introducing such features would naturally improve what counts as a valid email, currently the program simply checks whether the input has a “@” symbol. A possible feature is sharing bridges built identifying users by the username or their email,

### Performance

The FPS of the game varies wildly thought-out. Both PyBox2D and Pygame can use considerable amount of processing power. This game was created with in a virtual machine which would not have helped. The game has no allocation of graphics to a graphics card which would be considered in future development.

### General game ideas

Whilst finding a simple solution I avoided allowing the user to connected materials anywhere on other materials rather than just at joints; this is due to not being able to rotate rect objects, if I were to implement it, I would need to create an algorithm that checks the area between 2 points.

The strength of the materials is not realistic and the bridges that can currently be built are very strong, balancing the strengths will require a lot of testing, it would be suitable for an alpha version of the game to be released.

The feedback from the game to the user can be improved, this was an acceptable limitation for this project. There are several ways to implement such an idea: the joints darken in colour as they feel more stress and get closer to breaking, the user can hover the mouse above a joint and the stress the object is experiencing is displayed on screen.

### OOP design

Buttons, stick buttons and text box could have been designed better, they could have been subclasses to a superclass called box where the location and size of the box is defined, the characteristics of each would be defined in the subclass.

The vehicle class could have been designed better with more attributes such as density and vehicles length & height, the singular methods in each subclass could be condensed into the super class.

The rope and cable subclasses could have been made as subclasses under a superclass called chain which would be a subclass to material, this is because their methods almost identical.

# Appendix

## Testing screenshots

|  |  |  |
| --- | --- | --- |
| **Appendix number** | **Before action/input** | **After action/input** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  | eg |
| 15 |  |  |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |
| 20 |  |  |
| 21 |  |  |
| 22 |  |  |
| 23 |  |  |
| 24 |  |  |
| 25 |  |  |
| 26 |  |  |
| 27 |  |  |
| 28 |  |  |
| 29 |  |  |
| 30 |  |  |
| 31 |  |  |
| 32 |  |  |
| 33 |  |  |
| 34 |  |  |
| 35 |  |  |
| 36 |  |  |
| 37 |  |  |
| 38 |  |  |
| 39 |  |  |
| 40 |  |  |
| 41 |  |  |
| 42 |  |  |
| 43 |  |  |
| 44 |  |  |
| 45 |  |  |
| 46 |  |  |
| 47 |  |  |
| 48 |  |  |
| 49 |  |  |
| 50 |  |  |
| 51 |  |  |
| 52 |  |  |
| 53 |  |  |
| 54 |  |  |
| 55 |  |  |
| 56 |  |  |
| 57 |  |  |
| 58 |  |  |
| 59 |  |  |
| 60 |  |  |
| 61 |  |  |
| 62 |  |  |
| 63 |  |  |
| 64 |  |  |
| 65 |  |  |
| 66 |  |  |
| 67 |  |  |
| 68 |  |  |
| 69 |  |  |
| 70 |  |  |
| 71 |  |  |
| 72 |  |  |
| 73 |  |  |
| 74 |  |  |

|  |  |  |
| --- | --- | --- |
| 75 |  |  |

|  |  |  |
| --- | --- | --- |
| 76 |  |  |
| 77 |  |  |
| 78 |  |  |
| 79 |  |  |
| 80 |  |  |
| 81 |  |  |
| 82 |  |  |
| 83 |  |  |
| 84 |  |  |
| 85 |  |  |
| 86 |  |  |
| 87 |  |  |
| 88 |  |  |
| 89 |  |  |
| 90 |  |  |
| 91 |  |  |
| 92 |  |  |
| 93 |  |  |
| 94 |  |  |
| 95 |  |  |
| 96 |  |  |
| 97 |  |  |
| 98 |  |  |
| 99 |  |  |
| 100 |  |  |
| 101 |  |  |
| 102 |  |  |
| 103 |  |  |
| 104 |  |  |
| 105 |  |  |
| 106 |  |  |
| 107 |  |  |
| 108 |  |  |
| 109 |  |  |
| 110 |  |  |
| 111 |  |  |
| 112 |  |  |
| 113 |  |  |
| 114 |  |  |
| 115 |  |  |
| 116 |  |  |
| 117 |  |  |
| 118 |  |  |
| 119 |  |  |
| 120 |  |  |
| 121 |  |  |
| 122 |  |  |
| 123 |  |  |
| 124 |  |  |
| 125 |  |  |
| 126 |  |  |
| 127 |  |  |
| 128 |  |  |
| 129 |  |  |
| 130 |  |  |
| 131 |  |  |
| 132 |  |  |
| 133 |  |  |
| 134 |  |  |
| 135 |  |  |
| 136 |  |  |
| 137 |  |  |
| 138 |  |  |
| 139 |  |  |
| 140 |  |  |
| 141 |  |  |
| 142 |  |  |
| 143 |  |  |
| 144 |  |  |
| 145 |  |  |
| 146 |  |  |
| 147 |  |  |
| 148 |  |  |
| 149 |  |  |

## Adjacency List Structure

Three joints connected by three materials in a triangle shape.

{

'A': {

'location': (xA,yA),

'connectedJoints': [

{

'material': "MaterialA",

'joint': "B"

},

{

'material': "MaterialB",

'joint': "C"

}

]

},

'B' {

'location': (xB,yB),

'connectedJoints': [

{

'material': "MaterialA",

'joint': "A"

},

{

'material': "MaterialC",

'joint': "C"

}

]

},

'C' {

'location': (xC,yC),

'connectedJoints': [

{

'material': "MaterialB",

'joint': "A"

},

{

'material': "MaterialC",

'joint': "B"

}

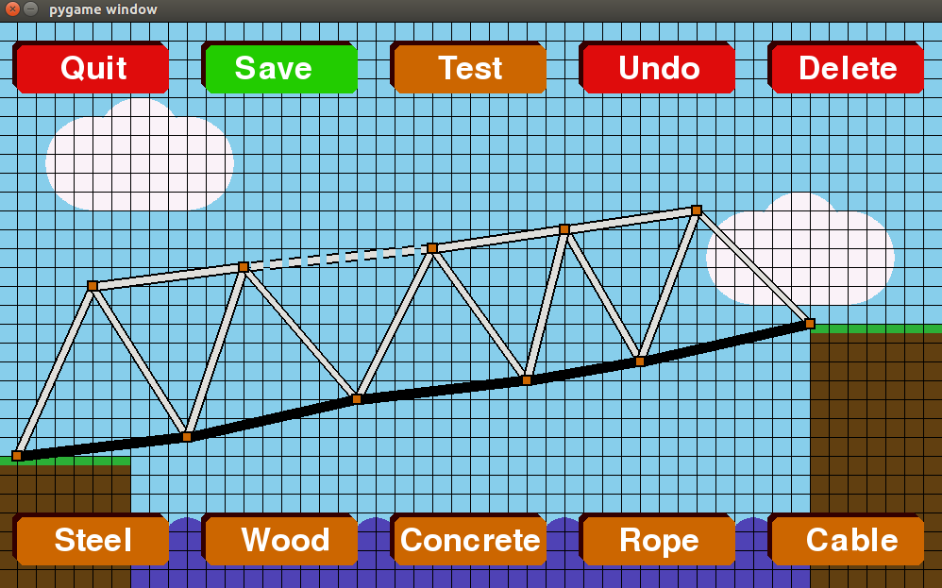
}

}

}

## Example Bridge File

{'21': {'location': (5, 16), 'connectedJoints': [{'material': 'Steel', 'joint': 15}, {'material': 'Steel', 'joint': 16}, {'material': 'Steel', 'joint': 22}]}, '15': {'location': (1, 7), 'connectedJoints': [{'material': 'Road', 'joint': 16}, {'material': 'Steel', 'joint': 21}]}, '22': {'location': (13, 17), 'connectedJoints': [{'material': 'Steel', 'joint': 16}, {'material': 'Steel', 'joint': 17}, {'material': 'Cable', 'joint': 23}, {'material': 'Steel', 'joint': 21}]}, '19': {'location': (34, 12), 'connectedJoints': [{'material': 'Road', 'joint': 18}, {'material': 'Road', 'joint': 20}, {'material': 'Steel', 'joint': 24}, {'material': 'Steel', 'joint': 25}]}, '16': {'location': (10, 8), 'connectedJoints': [{'material': 'Road', 'joint': 15}, {'material': 'Road', 'joint': 17}, {'material': 'Steel', 'joint': 21}, {'material': 'Steel', 'joint': 22}]}, '23': {'location': (23, 18), 'connectedJoints': [{'material': 'Steel', 'joint': 17}, {'material': 'Steel', 'joint': 18}, {'material': 'Steel', 'joint': 24}, {'material': 'Cable', 'joint': 22}]}, '24': {'location': (30, 19), 'connectedJoints': [{'material': 'Steel', 'joint': 18}, {'material': 'Steel', 'joint': 19}, {'material': 'Steel', 'joint': 25}, {'material': 'Steel', 'joint': 23}]}, '20': {'location': (43, 14), 'connectedJoints': [{'material': 'Road', 'joint': 19}, {'material': 'Steel', 'joint': 25}]}, '25': {'location': (37, 20), 'connectedJoints': [{'material': 'Steel', 'joint': 19}, {'material': 'Steel', 'joint': 20}, {'material': 'Steel', 'joint': 24}]}, '17': {'location': (19, 10), 'connectedJoints': [{'material': 'Road', 'joint': 16}, {'material': 'Road', 'joint': 18}, {'material': 'Steel', 'joint': 22}, {'material': 'Steel', 'joint': 23}]}, '18': {'location': (28, 11), 'connectedJoints': [{'material': 'Road', 'joint': 17}, {'material': 'Road', 'joint': 19}, {'material': 'Steel', 'joint': 23}, {'material': 'Steel', 'joint': 24}]}}



## References

* Al Sweigart - *“Python - Invent your own computer games with python”*
* [www.github.com/pybox2d/pybox2d](http://www.github.com/pybox2d/pybox2d)
* [www.stackoverflow.com](http://www.stackoverflow.com)
* [www.box2d.org](http://www.box2d.org)
* [www.w3schools.com](http://www.w3schools.com)
* P M Heathcote – “*AQA AS and A level Computer Science”*
* Kevin Roy Bond – “*A Level Computer Science for AQA unit 2*”

## Source code

### MAIN.py

import pygame,sys,Graphics,Classes,BridgeData,Build,Test,hashlib

from pygame.locals import \*

White = (255,255,255)

Red = (233,12,12)

LeafGreen = (97,138,61)

RoyalPurple = (120,81,169)

WarmBlue = (30,144,255)

BirchWhite = (248,223,161)

Pink = (255,105,180)

# produces a message on screen

def PopUpMessage(text,window):

center = (window.get\_rect().centerx, window.get\_rect().centery)

height = 50\*(len(text)+1)

message = Classes.TextBox(text,height,950,center,Red)

message.create(window)

# creates buttons for the options of what you can do with a bridge

def createButtonsbridgeOp(window,bridgeName,bridgeDate,buttons,error):

buttons['edit'].create(window)

buttons['test'].create(window)

buttons['delete'].create(window)

buttons['back5'].create(window)

cap = ("%s Last Edited: %s"%(bridgeName,bridgeDate))

height = 50

width = 600

x = window.get\_rect().centerx

y = 150

center = (x,y)

text = Classes.TextBox(cap,height,width,center)

text.create(window)

if error:

buttons['yes'].create(window)

buttons['cancel'].create(window)

PopUpMessage(["Are you sure you want to delete this bridge?"],window)

# creates the buttons for selecting an already made bridge

def createButtonsLoad(buttons,window,loadMenu,User\_ID):

font = pygame.font.SysFont(None,50)

listOfBridgeID = []

listOfBridgeNames = []

listOfBridgeDates = []

buttonsOnScreen = 0

numOfScreens = 0

upButton = False

downButton = False

results = BridgeData.getBridges(User\_ID)

start = loadMenu\*5

end = start + len(results)%5

if len(results) != 0:

if len(results)%5 != 0:

numOfScreens = len(results)//5 + 1

if loadMenu == numOfScreens - 1:

for bridge in range(start,end):

listOfBridgeID.append(results[bridge][0])

listOfBridgeNames.append(results[bridge][1].replace("'",""))

date = ("%s/%s/%s"%(results[bridge][2],results[bridge][3],results[bridge][4]))

listOfBridgeDates.append(date)

buttonsOnScreen += 1

else:

for bridge in range(start,start+5):

listOfBridgeID.append(results[bridge][0])

listOfBridgeNames.append(results[bridge][1].replace("'",""))

date = ("%s/%s/%s"%(results[bridge][2],results[bridge][3],results[bridge][4]))

listOfBridgeDates.append(date)

buttonsOnScreen += 1

else:

numOfScreens = len(results)//5

for bridge in range(start,start+5):

listOfBridgeID.append(results[bridge][0])

listOfBridgeNames.append(results[bridge][1].replace("'",""))

date = ("%s/%s/%s"%(results[bridge][2],results[bridge][3],results[bridge][4]))

listOfBridgeDates.append(date)

buttonsOnScreen += 1

else:

height = 100

width = 600

x = window.get\_rect().centerx

y = window.get\_rect().centery

center = (x,y)

text = Classes.TextBox("You have no bridges",height,width,center)

text.create(window)

if buttonsOnScreen >= 1:

buttons['bridge1'].create(window)

if buttonsOnScreen >= 2:

buttons['bridge2'].create(window)

if buttonsOnScreen >= 3:

buttons['bridge3'].create(window)

if buttonsOnScreen >= 4:

buttons['bridge4'].create(window)

if buttonsOnScreen == 5:

buttons['bridge5'].create(window)

if loadMenu != 0:

buttons['up'].create(window)

upButton = True

pygame.draw.line(window,White,(500,120),(480,140),10)

pygame.draw.line(window,White,(500,120),(520,140),10)

if loadMenu < numOfScreens-1:

buttons['down'].create(window)

downButton = True

pygame.draw.line(window,White,(500,500),(480,480),10)

pygame.draw.line(window,White,(500,500),(520,480),10)

buttons['back5'].create(window)

for bridgeName in range(0,len(listOfBridgeNames)):

text = font.render(listOfBridgeNames[bridgeName],True,(255,255,255),None)

textRect = text.get\_rect()

textRect.centerx = window.get\_rect().centerx

textRect.centery = 190 + bridgeName\*60

window.blit(text,textRect)

return [listOfBridgeID,listOfBridgeNames,listOfBridgeDates,buttonsOnScreen,upButton,downButton]

# creates the login and register buttons

def createButtonsInitial(buttons,window):

buttons['loginOp'].create(window)

buttons['registerOp'].create(window)

buttons['quit'].create(window)

# creates login screen

def createLogin(buttons,inputboxes,window,error):

buttons['login'].create(window)

buttons['back'].create(window)

if error == True:

buttons['ok'].create(window)

height = 250

width = 700

x = window.get\_rect().centerx

y = window.get\_rect().centery

center = (x,y)

text = Classes.TextBox("",height,width,center)

text.create(window)

inputboxes['Lusername'].create(window)

inputboxes['Lpassword'].create(window)

# creates register screen

def createRegister(buttons,inputboxes,window,error):

buttons['register'].create(window,error)

buttons['back2'].create(window,error)

if error == True:

buttons['ok'].create(window)

height = 400

width = 950

x = window.get\_rect().centerx

y = window.get\_rect().centery + 15

center = (x,y)

text = Classes.TextBox("",height,width,center)

text.create(window)

inputboxes['Rusername'].create(window)

inputboxes['Rpassword'].create(window)

inputboxes['RpasswordC'].create(window)

inputboxes['first'].create(window)

inputboxes['sec'].create(window)

inputboxes['email'].create(window)

# creates menu after logging in

def createButtonsMain(buttons,window):

buttons['play'].create(window)

buttons['instruc'].create(window)

buttons['logout'].create(window)

# creates play menu

def createButtonsPlay(buttons,window):

buttons['new'].create(window)

buttons['load'].create(window)

buttons['back3'].create(window)

# creates the menu for choosing the type of land they want to build on

def createButtonsLand(buttons,window):

buttons['flat'].create(window)

buttons['highlow'].create(window)

buttons['lowhigh'].create(window)

buttons['back5'].create(window)

def createButtonsDif(buttons,window):

buttons['easy'].create(window)

buttons['normal'].create(window)

buttons['hard'].create(window)

buttons['back5'].create(window)

# creates the menu to choose what vehicle the user wants to use

def createButtonsVehicle(buttons,window):

buttons['bike'].create(window)

buttons['car'].create(window)

buttons['truck'].create(window)

buttons['back5'].create(window)

# dispalys the instrucitons for the game

def createInstruc(buttons,window):

buttons['back4'].create(window)

cap = ['Your task is to build a stable bridge,', 'You can build it from a variety of materials,', 'and test it with either a bike, car or truck,', 'remember that your vehicle can only drive on','concrete!']

height = 300

width = 900

x = window.get\_rect().centerx

y = window.get\_rect().centery

center = (x,y)

text = Classes.TextBox(cap,height,width,center)

text.create(window)

# turns off input boxes when another is turned on

def DeActivateInputs(inputboxes,current):

for box in inputboxes:

if box != current:

inputboxes[box].deActive()

# checks user input for logging in and checks if the details are correct

def checkLogin(inputboxes):

checkList = [" ", ";", "=", "'", '"']

error = []

User\_ID = ""

emptyCount = 0

invalidCharCount = 0

for box in inputboxes:

if (box == "Lusername") or (box == "Lpassword"):

for char in checkList:

if inputboxes[box].getCap() == "":

emptyCount += 1

elif char in inputboxes[box].getCap():

invalidCharCount += 1

if emptyCount > 0:

text = "All fields must be entered"

error.append(text)

if invalidCharCount > 0:

text = "Incorrect details"

error.append(text)

if error == []:

Hash = hashlib.sha512()

username = inputboxes['Lusername'].getCap()

passCap = bytes(inputboxes['Lpassword'].getCap(), encoding='utf-8')

Hash.update(passCap)

password = Hash.hexdigest()

connect = BridgeData.findUser(username,password)

if connect == "Not":

error.append("Incorrect details")

elif connect == "Error":

error.append("Error connecting to database")

else:

User\_ID = BridgeData.getUser\_ID(username)

if User\_ID == "":

error.append("Error connecting to the database")

return error,User\_ID

# checks user input for registering, if inputs are valid it will register the user

def checkRegister(inputboxes):

checkList = [" ", ";", "=", "'", '"']

error = []

emptyCount = 0

for box in inputboxes:

if (box != "Lusername") and (box != "Lpassword"):

for char in checkList:

if (inputboxes[box].getCap() == "") or (char in inputboxes[box].getCap()):

emptyCount += 1

if emptyCount > 0:

text = "All fields must be entered and cannot contain ;, =, ' or \""

error.append(text)

if (not("@" in inputboxes['email'].getCap())) and (inputboxes['email'].getCap() != ""):

text = "Please enter a valid email"

error.append(text)

if inputboxes['Rpassword'].getCap() != inputboxes['RpasswordC'].getCap():

text = "Passwords do not match"

error.append(text)

if error == []:

username = inputboxes['Rusername'].getCap()

UsernameState = BridgeData.findUsername(username)

if UsernameState != "":

if UsernameState == "Taken":

error.append("Username taken")

elif UsernameState == "Error":

error.append("Error connecting to database")

email = inputboxes['email'].getCap()

EmailState = BridgeData.findEmail(email)

if EmailState != "":

if EmailState == "Taken":

error.append("Email taken")

elif EmailState == "Error":

error.append("Error connecting to database")

if error == []:

Hash = hashlib.sha512()

first = inputboxes['first'].getCap()

sec = inputboxes['sec'].getCap()

email = inputboxes['email'].getCap()

username = inputboxes['Rusername'].getCap()

passCap = bytes(inputboxes['Rpassword'].getCap(), encoding='utf-8')

Hash.update(passCap)

password = Hash.hexdigest()

connect = BridgeData.addUser(first,sec,email,username,password)

if connect != "":

error.append("Error connecting to database")

return error

# creates buttons and handles clicks and typing

def menuLoop(Menu,window,info):

User\_ID = info['User\_ID']

bridgeID = info['bridgeID']

buttons = {

'loginOp':Classes.Button(' Login ',window.get\_rect().centerx,250),

'registerOp':Classes.Button(' Register ',window.get\_rect().centerx,350),

'quit':Classes.Button(' Quit ',window.get\_rect().centerx,450),

'login':Classes.Button(' Login ',650,500),

'back':Classes.Button(' Back ',350,500),

'register':Classes.Button(' Register ',650,555),

'back2': Classes.Button(' Back ',350,555),

'play':Classes.Button(' Play ',window.get\_rect().centerx,250),

'instruc':Classes.Button(' Instructions ',window.get\_rect().centerx,350),

'logout':Classes.Button(' Logout ',window.get\_rect().centerx,450),

'new':Classes.Button(' New Bridge ',window.get\_rect().centerx,250),

'load':Classes.Button(' Load Bridge ',window.get\_rect().centerx,350),

'back3':Classes.Button(' Back ',window.get\_rect().centerx,450),

'back4':Classes.Button(' Back ',window.get\_rect().centerx,550),

'flat':Classes.Button(' Flat ',500,450),

'highlow':Classes.Button(' High to Low ',250,450),

'lowhigh':Classes.Button(' Low to High ',750,450),

'back5':Classes.Button(' Back ',500,550),

'normal':Classes.Button(' Normal ',500,450),

'easy':Classes.Button(' Easy ',250,450),

'hard':Classes.Button(' Hard ',750,450),

'ok':Classes.Button(' OK ',500,550,Red),

'bridge1':Classes.Button(' ',500,190,LeafGreen),

'bridge2':Classes.Button(' ',500,250,RoyalPurple),

'bridge3':Classes.Button(' ',500,310,WarmBlue),

'bridge4':Classes.Button(' ',500,370,BirchWhite),

'bridge5':Classes.Button(' ',500,430,Pink),

'up':Classes.Button(' ',500,130,width=30,height=15),

'down':Classes.Button(' ',500,490,width=30,height=15),

'edit':Classes.Button(' Edit ',500,250),

'test':Classes.Button(' Test ',500,350),

'delete':Classes.Button(' Delete ',500,450,Red),

'yes':Classes.Button(' Yes ',700,500,Red),

'cancel':Classes.Button(' Cancel ',300,500,Red),

'bike':Classes.Button(' Bike ',250,450),

'car':Classes.Button(' Car ',500,450),

'truck':Classes.Button(' Truck ',750,450),

}

inputboxes = {

'Lusername':Classes.InputBox("Username:",365,200,470,60),

'Lpassword':Classes.InputBox("Password:",365,300,470,60),

'first':Classes.InputBox("First Name:",365,123,470,60),

'sec':Classes.InputBox("Surname:",365,188,470,60),

'email':Classes.InputBox("Email:",150,253,810,60),

'Rusername':Classes.InputBox("Username:",365,318,470,60),

'Rpassword':Classes.InputBox("Password:",365,383,470,60),

'RpasswordC':Classes.InputBox("Re-Type:",365,448,470,60)

}

menu = Menu

loadMenu = 0

#clock = pygame.time.Clock()

click = False

Type = False

error = False

loadBridge = False

vehicle = ""

dif = ""

landscape= ""

while menu != "build" and menu != "test":

for event in pygame.event.get():

if event.type == QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

click = True

elif event.type == pygame.KEYDOWN:

Type = True

if event.key == pygame.K\_BACKSPACE:

char = 'back'

else:

char = event.unicode

Graphics.BackDrop("","",window)

if menu == "initial":

createButtonsInitial(buttons,window)

elif menu == "login":

createLogin(buttons,inputboxes,window,error)

if error == True:

PopUpMessage(sentences,window)

elif menu == "register":

createRegister(buttons,inputboxes,window,error)

if error == True:

PopUpMessage(sentences,window)

elif menu == "main":

createButtonsMain(buttons,window)

elif menu == "sec":

createButtonsPlay(buttons,window)

elif menu == "instruc":

createInstruc(buttons,window)

elif menu == "LandOp":

Graphics.LandImage(window)

createButtonsLand(buttons,window)

elif menu == "Dif":

Graphics.DifImage(window,landscape)

createButtonsDif(buttons,window)

elif menu == "load":

listOfBridgeID,listOfBridgeNames,listOfBridgeDates,buttonsOnScreen,upButton,downButton = createButtonsLoad(buttons,window,loadMenu,User\_ID)

elif menu == "bridgeOp":

createButtonsbridgeOp(window,bridgeName,bridgeDate,buttons,error)

elif menu == "chooseVehicle":

Graphics.vehicleImage(window)

createButtonsVehicle(buttons,window)

if click == True:

if menu == 'initial':

if buttons['loginOp'].ifClick():

menu = 'login'

elif buttons['registerOp'].ifClick():

menu = 'register'

elif buttons['quit'].ifClick():

pygame.quit()

sys.exit()

elif menu == 'login':

if error == False:

if inputboxes['Lusername'].ifClick():

inputboxes['Lusername'].makeActive()

DeActivateInputs(inputboxes,"Lusername")

elif inputboxes['Lpassword'].ifClick():

inputboxes['Lpassword'].makeActive()

DeActivateInputs(inputboxes,"Lpassword")

elif buttons['login'].ifClick():

sentences,User\_ID = checkLogin(inputboxes)

if sentences != []:

error = True

else:

error = False

menu = "main"

DeActivateInputs(inputboxes," ")

elif buttons['back'].ifClick():

menu = 'initial'

DeActivateInputs(inputboxes," ")

else:

DeActivateInputs(inputboxes," ")

else:

if buttons['ok'].ifClick():

error = False

elif menu == 'register':

if error == False:

if inputboxes['first'].ifClick():

inputboxes['first'].makeActive()

DeActivateInputs(inputboxes,"first")

elif inputboxes['sec'].ifClick():

inputboxes['sec'].makeActive()

DeActivateInputs(inputboxes,"sec")

elif inputboxes['email'].ifClick():

inputboxes['email'].makeActive()

DeActivateInputs(inputboxes,"email")

elif inputboxes['Rusername'].ifClick():

inputboxes['Rusername'].makeActive()

DeActivateInputs(inputboxes,"Rusername")

elif inputboxes['Rpassword'].ifClick():

inputboxes['Rpassword'].makeActive()

DeActivateInputs(inputboxes,"Rpassword")

elif inputboxes['RpasswordC'].ifClick():

inputboxes['RpasswordC'].makeActive()

DeActivateInputs(inputboxes,"RpasswordC")

elif buttons['register'].ifClick():

sentences = checkRegister(inputboxes)

if sentences != []:

error = True

else:

error = False

menu = 'login'

DeActivateInputs(inputboxes," ")

elif buttons['back2'].ifClick():

menu = 'initial'

DeActivateInputs(inputboxes," ")

else:

DeActivateInputs(inputboxes," ")

else:

if buttons['ok'].ifClick():

error = False

elif menu == 'main':

if buttons['play'].ifClick():

menu = 'sec'

elif buttons['instruc'].ifClick():

menu = 'instruc'

elif buttons['logout'].ifClick():

menu = 'initial'

elif menu == "sec":

if buttons['new'].ifClick():

menu = "LandOp"

elif buttons['load'].ifClick():

menu = "load"

elif buttons['back3'].ifClick():

menu = "main"

elif menu == "instruc":

if buttons['back4'].ifClick():

menu = "main"

elif menu == "LandOp":

if buttons['flat'].ifClick():

landscape = 1

menu = "Dif"

elif buttons['highlow'].ifClick():

landscape = 2

menu = "Dif"

elif buttons['lowhigh'].ifClick():

menu = "Dif"

landscape = 3

elif buttons['back5'].ifClick():

menu = "sec"

elif menu == "Dif":

if buttons['normal'].ifClick():

dif = "normal"

menu = "build"

loadBridge = False

elif buttons['easy'].ifClick():

dif = "easy"

menu = "build"

loadBridge = False

elif buttons['hard'].ifClick():

menu = "build"

dif = "hard"

loadBridge = False

elif buttons['back5'].ifClick():

menu = "LandOp"

elif menu == "load":

bridgeOp = 0

if buttonsOnScreen >= 1:

if buttons['bridge1'].ifClick():

menu = "bridgeOp"

bridgeOp = 1

if buttonsOnScreen >= 2:

if buttons['bridge2'].ifClick():

menu = "bridgeOp"

bridgeOp = 2

if buttonsOnScreen >= 3:

if buttons['bridge3'].ifClick():

menu = "bridgeOp"

bridgeOp = 3

if buttonsOnScreen >= 4:

if buttons['bridge4'].ifClick():

menu = "bridgeOp"

bridgeOp = 4

if buttonsOnScreen == 5:

if buttons['bridge5'].ifClick():

menu = "bridgeOp"

bridgeOp = 5

if bridgeOp != 0:

bridgeID = listOfBridgeID[bridgeOp-1]

bridgeName = listOfBridgeNames[bridgeOp-1]

bridgeDate = listOfBridgeDates[bridgeOp-1]

if upButton:

if buttons['up'].ifClick():

loadMenu -= 1

if downButton:

if buttons['down'].ifClick():

loadMenu += 1

if buttons['back5'].ifClick():

menu = "sec"

elif menu == "bridgeOp":

if error == False:

if buttons['edit'].ifClick():

menu = "build"

loadBridge = True

dif = ""

landscape = ""

elif buttons['test'].ifClick():

menu = "chooseVehicle"

elif buttons['delete'].ifClick():

error = True

elif buttons['back5'].ifClick():

menu = "load"

else:

if buttons['yes'].ifClick():

BridgeData.deleteBridge(bridgeID)

menu = "load"

error = False

loadMenu = 0

elif buttons['cancel'].ifClick():

error = False

elif menu == "chooseVehicle":

if buttons['bike'].ifClick():

menu = "test"

vehicle = "bike"

elif buttons['car'].ifClick():

menu = "test"

vehicle = "car"

elif buttons['truck'].ifClick():

menu = "test"

vehicle = "truck"

elif buttons['back5'].ifClick():

menu = "load"

click = False

if Type == True:

if menu == 'login':

if inputboxes['Lusername'].getActive() == True:

if char == 'back':

inputboxes['Lusername'].Back()

elif (char != 'back') and (inputboxes['Lusername'].getCapLength() < 20):

inputboxes['Lusername'].Add(char)

elif inputboxes['Lpassword'].getActive() == True:

if char == 'back':

inputboxes['Lpassword'].Back()

elif (char != 'back') and (inputboxes['Lpassword'].getCapLength() < 20):

inputboxes['Lpassword'].Add("X")

elif menu == 'register':

if inputboxes['first'].getActive() == True:

if char == 'back':

inputboxes['first'].Back()

elif (char != 'back') and (inputboxes['first'].getCapLength() < 20):

inputboxes['first'].Add(char)

elif inputboxes['sec'].getActive() == True:

if char == 'back':

inputboxes['sec'].Back()

elif (char != 'back') and (inputboxes['sec'].getCapLength() < 20):

inputboxes['sec'].Add(char)

elif inputboxes['email'].getActive() == True:

if char == 'back':

inputboxes['email'].Back()

elif (char != 'back') and (inputboxes['email'].getCapLength() < 50):

inputboxes['email'].Add(char)

elif inputboxes['Rusername'].getActive() == True:

if char == 'back':

inputboxes['Rusername'].Back()

elif (char != 'back') and (inputboxes['Rusername'].getCapLength() < 20):

inputboxes['Rusername'].Add(char)

elif inputboxes['Rpassword'].getActive() == True:

if char == 'back':

inputboxes['Rpassword'].Back()

elif (char != 'back') and (inputboxes['Rpassword'].getCapLength() < 20):

inputboxes['Rpassword'].Add("X")

elif inputboxes['RpasswordC'].getActive() == True:

if char == 'back':

inputboxes['RpasswordC'].Back()

elif (char != 'back') and (inputboxes['RpasswordC'].getCapLength() < 20):

inputboxes['RpasswordC'].Add("X")

Type = False

char = ""

#clock.tick()

#fps = clock.get\_fps()

#print(fps)

pygame.display.update()

info = {'User\_ID':User\_ID, 'dif':dif,'land':landscape,'bridgeID':bridgeID, 'loadBridge':loadBridge,'build':False,'test':False,'vehicle':vehicle}

if menu == "build":

info['build'] = True

elif menu == "test":

info['test'] = True

return info

# initial function, handles switching between files

def Main():

pygame.init()

window = pygame.display.set\_mode((1000,600),0,32)

Next = "initial"

info = {'User\_ID':0,'bridgeID':0,'build':False,'test':False}

while True:

if info['build'] == False and info['test'] == False:

info = menuLoop(Next,window,info)

if info['build']:

Next,info = Build.Main(info,window)

if info['test']:

Next,info = Test.Main(info,window)

Main()

### Classes.py

This file is exaplained in more detail in OOP Classes on page 25

from Box2D import \*

import pygame

import Graphics

import math

WHITE = (255,255,255)

LBROWN = (204,102,0)

DBROWN = (51,0,0)

class TextBox():

def \_\_init\_\_(self,cap,height,width,center,colour1=LBROWN,colour2=DBROWN):

self.cap = cap

self.height = height

self.width = width

self.center = center

self.colour1 = colour1

self.colour2 = colour2

def create(self,window):

word = self.cap

font = pygame.font.SysFont(None,50)

height = self.height / 2

width = self.width / 2

x,y = self.center

# each element in the list will be on a seperate line

if type(word) == str:

temp = word

word = []

word.append(temp)

# defines vertices

poly1 = ((x-width,y+(height-5)),(x-(width-5),y+height),(x+(width-5),y+height),(x+width,y+(height-5)),(x+width,y-(height-5)),(x+(width-5),y-height),(x-(width-5),y-height),(x-width,y-(height-5)))

poly2 = ((x-(width+5),y+(height-10)),(x-width,y+(height-5)),(x+(width-10),y+(height-5)),(x+(width-5),y+(height-10)),(x+(width-5),y-height),(x+(width-10),y-(height+5)),(x-width,y-(height+5)),(x-(width+5),y-height))

pygame.draw.polygon(window,self.colour2,poly2)

pygame.draw.polygon(window,self.colour1,poly1)

# pixel location

if (len(word))%2 == 0:

start = 25+((-50)\*(len(word)/2))

end = abs(start)+25

else:

start = (-50)\*((len(word)-1)/2)

end = abs(start)+50

wordTrack = 0

for loc in range(int(start), int(end), 50):

Text = font.render(word[wordTrack],True,WHITE,None)

textRect = Text.get\_rect()

textRect.centerx = x

textRect.centery = y + loc

window.blit(Text,textRect)

wordTrack += 1

class InputBox():

def \_\_init\_\_(self,name,xCo,yCo,width,height):

self.name = name

self.cap = ""

self.yCo = yCo

self.xCo = xCo

self.rect = pygame.Rect(xCo,yCo,width,height)

self.active = False

def create(self,window):

font = pygame.font.SysFont(None, 50)

xCo = self.xCo

yCo = self.yCo

text1 = font.render(self.cap,True,WHITE,None)

textRect1 = text1.get\_rect()

text2 = font.render(self.name,True,WHITE,None)

textRect2 = text2.get\_rect()

textRect1.left = xCo + 10

textRect1.top = yCo + 10

textRect2.right = self.rect.left - 10

textRect2.top = self.rect.top + 10

if self.active:

colour = LBROWN

else:

colour = DBROWN

pygame.draw.rect(window,colour,self.rect)

window.blit(text1,textRect1)

window.blit(text2,textRect2)

def ifClick(self):

if self.rect.collidepoint(pygame.mouse.get\_pos()) == True:

return True

else:

return False

def Add(self, char):

self.cap += char

def makeActive(self):

self.active = True

def deActive(self):

self.active = False

def getActive(self):

active = self.active

return active

def Back(self):

self.cap = self.cap[:-1]

def getCapLength(self):

length = len(self.cap)

return length

def getCap(self):

cap = self.cap

return cap

class Button():

def \_\_init\_\_(self,cap,locx,locy,colour1=LBROWN,colour2=DBROWN,width=105,height=20):

self.cap = cap

self.locy = locy

self.locx = locx

self.rect = 0

self.colour1 = colour1

self.colour2 = colour2

self.width = width

self.height = height

def create(self,window,error=False):

word = self.cap

font = pygame.font.SysFont(None, 50)

locy = self.locy

locx = self.locx

width = self.width

height = self.height

text = font.render(word,True,WHITE,None)

textRect = text.get\_rect()

self.rect = text.get\_rect()

textRect.centerx = locx

textRect.centery = locy

self.rect.centerx = locx

self.rect.centery = locy

x = textRect.centerx

y = textRect.centery

# defines vertices

poly1 = ((x-(width+5),y+height),(x-width,y+height+5),(x+width,y+height+5),(x+width+5,y+height),(x+width+5,y-height),(x+width,y-(height+5)),(x-width,y-(height+5)),(x-(width+5),y-height))

poly2 = ((x-(width+10),y+height-5),(x-(width+5),y+height),(x+width-5,y+height),(x+width,y+height-5),(x+width,y-(height+5)),(x+width-5,y-(height+10)),(x-(width+5),y-(height+10)),(x-(width+10),y-(height+5)))

if textRect.collidepoint(pygame.mouse.get\_pos()) and error == False:

pygame.draw.polygon(window,self.colour1,poly2)

pygame.draw.polygon(window,self.colour2,poly1)

else:

pygame.draw.polygon(window,self.colour2,poly2)

pygame.draw.polygon(window,self.colour1,poly1)

window.blit(text,textRect)

def ifClick(self):

if self.rect.collidepoint(pygame.mouse.get\_pos()) == True:

return True

else:

return False

class stickButton(Button):

def \_\_init\_\_(self,cap,locx,locy,colour1=LBROWN,colour2=DBROWN):

Button.\_\_init\_\_(self,cap,locx,locy,colour1,colour2)

self.On = False

def create(self,window):

word = self.cap

font = pygame.font.SysFont(None, 50)

locy = self.locy

locx = self.locx

text = font.render(word,True,WHITE,None)

textRect = text.get\_rect()

self.rect = text.get\_rect()

textRect.centerx = locx

textRect.centery = locy

self.rect.centerx = locx

self.rect.centery = locy

x = textRect.centerx

y = textRect.centery

# defines vertices

poly1 = ((x-80,y+20),(x-75,y+25),(x+75,y+25),(x+80,y+20),(x+80,y-20),(x+75,y-25),(x-75,y-25),(x-80,y-20))

poly2 = ((x-85,y+15),(x-80,y+20),(x+70,y+20),(x+75,y+15),(x+75,y-25),(x+70,y-30),(x-80,y-30),(x-85,y-25))

if self.On:

pygame.draw.polygon(window,self.colour1,poly2)

pygame.draw.polygon(window,self.colour2,poly1)

else:

pygame.draw.polygon(window,self.colour2,poly2)

pygame.draw.polygon(window,self.colour1,poly1)

window.blit(text,textRect)

def ifClick(self):

if self.rect.collidepoint(pygame.mouse.get\_pos()) == True:

return True

else:

return False

def getOn(self):

ifOn = self.On

return ifOn

def turnOn(self):

self.On = True

def turnOff(self):

self.On = False

class Material():

def \_\_init\_\_(self):

self.x1 = 0

self.x2 = 0

self.y1 = 0

self.y2 = 0

self.joint1 = 0

self.joint2 = 0

self.Colour = 0

self.error = False

self.material = ""

self.body = ""

self.groupIndex = -2

self.density = 1

self.B2joints1 = []

self.B2joints2 = []

self.maxForce = 0

def getMaxForce(self):

maxForce = self.maxForce

return maxForce

def ifBreak(self,world,timeStep):

totalX = 0

totalY = 0

numOfJoints = len(self.B2joints1) + len(self.B2joints2)

maxForce = 0

for joint in self.B2joints1:

x,y = joint['reference'].GetReactionForce(1/timeStep)

totalX += x

totalY += y

maxForce += joint['maxForce']

reaction1 = (totalX\*\*2 + totalY\*\*2)\*\*0.5

totalX = 0

totalY = 0

for joint in self.B2joints2:

x,y = joint['reference'].GetReactionForce(1/timeStep)

totalX += x

totalY += y

maxForce += joint['maxForce']

reaction2 = (totalX\*\*2 + totalY\*\*2)\*\*0.5

maxForce = (maxForce + self.maxForce) / (numOfJoints + 1)

if reaction1 + reaction2 > maxForce:

if reaction1 > reaction2:

for joint in self.B2joints1:

world.DestroyJoint(joint['reference'])

del self.B2joints1[:]

else:

for joint in self.B2joints2:

world.DestroyJoint(joint['reference'])

del self.B2joints2[:]

def addB2joints2(self,joint):

self.B2joints2.append(joint)

def addB2joints1(self,joint):

self.B2joints1.append(joint)

def getBody(self):

body = self.body

return body

def getMaterial(self):

material = self.material

return material

def errorTrue(self):

self.error = True

def getJoint1(self):

joint = self.joint1

return joint

def getJoint2(self):

joint = self.joint2

return joint

def getError(self):

error = self.error

return error

def getStart(self):

pos = (self.x1,self.y1)

return pos

def getEnd(self):

pos = (self.x2,self.y2)

return pos

def setjoint1(self,n):

self.joint1 = n

def setjoint2(self,n):

self.joint2 = n

def checkPlacement(self):

if ((self.x1 == self.x2) and (self.y1 == self.y2)) or (((((self.x2-self.x1)\*\*2)+((self.y2-self.y1)\*\*2))\*\*0.5)>200):

self.error = True

else:

self.error = False

def setCo1(self,x,y):

self.x1 = x

self.y1 = y

def setCo2(self,x,y):

self.x2 = x

self.y2 = y

def updateLocation(self,x,y):

self.x2 = x

self.y2 = y

def Draw(self,window):

pygame.draw.line(window,(0,0,0),(self.x1,self.y1),(self.x2,self.y2),11)

if self.error == False:

pygame.draw.line(window,self.Colour,(self.x1,self.y1),(self.x2,self.y2),7)

else:

pygame.draw.line(window,(233,12,12),(self.x1,self.y1),(self.x2,self.y2),7)

def createBody(self,world):

x1 = self.x1 / 20

x2 = self.x2 / 20

y1 = 30 - self.y1/20

y2 = 30 - self.y2/20

deltax = x2 - x1

deltay = y2 - y1

length = (deltay\*\*2 + deltax\*\*2)\*\*0.5

angle = math.atan2(deltay,deltax)

pos = (x1+deltax/2,y1+deltay/2)

fix = b2FixtureDef(shape=b2PolygonShape(box=(length/2,0.2)),density=self.density,friction=0.2,categoryBits=0x0002,maskBits=0x0004)

fix.filter.groupIndex = self.groupIndex

self.body = world.CreateDynamicBody(position=pos,fixtures=fix,angle=angle)

def testDraw(self,window):

posX,posY = self.body.position

posX \*= 20

posY = 600 - posY\*20

angle = 2\*math.pi - self.body.angle

radius = (((self.x2-self.x1)\*\*2+(self.y2-self.y1)\*\*2)\*\*0.5)/2

self.x1 = posX - radius\*math.cos(angle)

self.y1 = posY - radius\*math.sin(angle)

self.x2 = posX + radius\*math.cos(angle)

self.y2 = posY + radius\*math.sin(angle)

self.Draw(window)

class Steel(Material):

def \_\_init\_\_(self):

Material.\_\_init\_\_(self)

self.Colour = (224,223,219)

self.material = "Steel"

self.density = 8000

self.maxForce = 4000000

class Wood(Material):

def \_\_init\_\_(self):

Material.\_\_init\_\_(self)

self.Colour = (120,81,45)

self.material = "Wood"

self.density = 2000

self.maxForce = 3000000

class Road(Material):

def \_\_init\_\_(self):

Material.\_\_init\_\_(self)

self.Colour = (0,0,0)

self.material = "Road"

self.groupIndex = 1

self.density = 3000

self.maxForce = 2000000

class Rope(Material):

def \_\_init\_\_(self):

Material.\_\_init\_\_(self)

self.Colour = (120,81,45)

self.material = "Rope"

self.density = 500

self.body = []

self.chainJoints = []

self.maxForce = 2000000

def ifBreak(self,world,timeStep):

Material.ifBreak(self,world,timeStep)

toDelete = -1

for joint in range(0,len(self.chainJoints)):

forceX,forceY = self.chainJoints[joint].GetReactionForce(1/timeStep)

force = (forceX\*\*2 + forceY\*\*2)\*\*0.5

if force > self.maxForce:

toDelete = joint

if toDelete != -1:

world.DestroyJoint(self.chainJoints[toDelete])

del self.chainJoints[toDelete]

def checkPlacement(self):

if ((self.x1 == self.x2) and (self.y1 == self.y2)) or (((((self.x2-self.x1)\*\*2)+((self.y2-self.y1)\*\*2))\*\*0.5)>400):

self.error = True

else:

self.error = False

def Draw(self,window):

xChange = self.x2 - self.x1

yChange = self.y2 - self.y1

for i in range(0,10):

xPoint1 = self.x1 + (i\*0.1\*xChange)

yPoint1 = self.y1 + (i\*0.1\*yChange)

xPoint2 = xPoint1 + (0.06\*xChange)

yPoint2 = yPoint1 + (0.06\*yChange)

pygame.draw.line(window,(0,0,0),(xPoint1,yPoint1),(xPoint2,yPoint2),11)

if self.error == False:

pygame.draw.line(window,self.Colour,(xPoint1,yPoint1),(xPoint2,yPoint2),7)

else:

pygame.draw.line(window,(233,12,12),(xPoint1,yPoint1),(xPoint2,yPoint2),7)

def createBody(self,world):

x1 = self.x1 / 20

x2 = self.x2 / 20

y1 = 30 - self.y1/20

y2 = 30 - self.y2/20

deltax = x2 - x1

deltay = y2 - y1

length = (deltay\*\*2 + deltax\*\*2)\*\*0.5

angle = math.atan2(deltay,deltax)

stepx = deltax/10

stepy = deltay/10

fix = b2FixtureDef(shape=b2PolygonShape(box=(length/20,0.35)),density=self.density,friction=0.2,categoryBits=0x0002,maskBits=0x0004)

fix.filter.groupIndex = self.groupIndex

prevBody = ""

for i in range(0,10):

pos = (x1 + (i+0.5)\*stepx, y1 + (i+0.5)\*stepy)

body = world.CreateDynamicBody(position=pos,fixtures = fix,angle = angle)

if i != 0:

joint = world.CreateRevoluteJoint(bodyA=prevBody,bodyB=body,anchor=(x1 + i\*stepx, y1 + i\*stepy))

self.chainJoints.append(joint)

self.body.append(body)

prevBody = body

def testDraw(self,window):

radius = (((self.x2-self.x1)\*\*2+(self.y2-self.y1)\*\*2)\*\*0.5)/20

for body in self.body:

posX,posY = body.position

posX \*= 20

posY = 600 - posY\*20

angle = 2\*math.pi - body.angle

point1 = (posX - radius\*math.cos(angle),posY - radius\*math.sin(angle))

point2 = (posX + radius\*math.cos(angle),posY + radius\*math.sin(angle))

pygame.draw.line(window,(0,0,0),point1,point2,11)

pygame.draw.line(window,self.Colour,point1,point2,7)

class Cable(Material):

def \_\_init\_\_(self):

Material.\_\_init\_\_(self)

self.Colour = (224,223,219)

self.material = "Cable"

self.density = 1500

self.body = []

self.chainJoints = []

self.maxForce = 3000000

def ifBreak(self,world,timeStep):

Material.ifBreak(self,world,timeStep)

toDelete = -1

for joint in range(0,len(self.chainJoints)):

forceX,forceY = self.chainJoints[joint].GetReactionForce(1/timeStep)

force = (forceX\*\*2 + forceY\*\*2)\*\*0.5

if force > self.maxForce:

toDelete = joint

if toDelete != -1:

world.DestroyJoint(self.chainJoints[toDelete])

del self.chainJoints[toDelete]

def checkPlacement(self):

if ((self.x1 == self.x2) and (self.y1 == self.y2)) or (((((self.x2-self.x1)\*\*2)+((self.y2-self.y1)\*\*2))\*\*0.5)>400):

self.error = True

else:

self.error = False

def Draw(self,window):

xChange = self.x2 - self.x1

yChange = self.y2 - self.y1

for i in range(0,10):

xPoint1 = self.x1 + (i\*0.1\*xChange)

yPoint1 = self.y1 + (i\*0.1\*yChange)

xPoint2 = xPoint1 + (0.06\*xChange)

yPoint2 = yPoint1 + (0.06\*yChange)

pygame.draw.line(window,(0,0,0),(xPoint1,yPoint1),(xPoint2,yPoint2),11)

if self.error == False:

pygame.draw.line(window,self.Colour,(xPoint1,yPoint1),(xPoint2,yPoint2),7)

else:

pygame.draw.line(window,(233,12,12),(xPoint1,yPoint1),(xPoint2,yPoint2),7)

def createBody(self,world):

x1 = self.x1 / 20

x2 = self.x2 / 20

y1 = 30 - self.y1/20

y2 = 30 - self.y2/20

deltax = x2 - x1

deltay = y2 - y1

length = (deltay\*\*2 + deltax\*\*2)\*\*0.5

angle = math.atan2(deltay,deltax)

stepx = deltax/10

stepy = deltay/10

fix = b2FixtureDef(shape=b2PolygonShape(box=(length/20,0.35)),density=self.density,friction=0.2,categoryBits=0x0002,maskBits=0x0004)

fix.filter.groupIndex = self.groupIndex

prevBody = ""

for i in range(0,10):

pos = (x1 + (i+0.5)\*stepx, y1 + (i+0.5)\*stepy)

body = world.CreateDynamicBody(position=pos,fixtures = fix,angle = angle)

if i != 0:

joint = world.CreateRevoluteJoint(bodyA=prevBody,bodyB=body,anchor=(x1 + i\*stepx, y1 + i\*stepy))

self.chainJoints.append(joint)

self.body.append(body)

prevBody = body

def testDraw(self,window):

radius = (((self.x2-self.x1)\*\*2+(self.y2-self.y1)\*\*2)\*\*0.5)/20

for body in self.body:

posX,posY = body.position

posX \*= 20

posY = 600 - posY\*20

angle = 2\*math.pi - body.angle

point1 = (posX - radius\*math.cos(angle),posY - radius\*math.sin(angle))

point2 = (posX + radius\*math.cos(angle),posY + radius\*math.sin(angle))

pygame.draw.line(window,(0,0,0),point1,point2,11)

pygame.draw.line(window,self.Colour,point1,point2,7)

class vehicle():

def \_\_init\_\_(self):

self.chassis = ""

self.wheel1 = ""

self.wheel2 = ""

self.joint1 = ""

self.joint2 = ""

def forward(self):

self.joint1.motorSpeed = -30

self.joint2.motorSpeed = -30

def back(self):

self.joint1.motorSpeed = 30

self.joint2.motorSpeed = 30

def stop(self):

self.joint1.motorSpeed = 0

self.joint2.motorSpeed = 0

class car(vehicle):

def create(self,world,height):

height += 3

chassisDef = b2FixtureDef(shape=b2PolygonShape(box=(1.5,0.8)),density=500,friction=0.3,categoryBits=0x0004,maskBits=0x0002)

chassisDef.filter.groupIndex = -2

self.chassis = world.CreateDynamicBody(position=(4,height),fixtures=chassisDef)

wheel1Def = b2FixtureDef(shape=b2CircleShape(radius=0.8),density=100,categoryBits=0x0004,maskBits=0x0002)

wheel1Def.filter.groupIndex = -2

self.wheel1 = world.CreateDynamicBody(position=(5.5,height-0.8),fixtures=wheel1Def)

wheel2Def = b2FixtureDef(shape=b2CircleShape(radius=0.8),density=100,categoryBits=0x0004,maskBits=0x0002)

wheel2Def.filter.groupIndex = -2

self.wheel2 = world.CreateDynamicBody(position=(2.5,height-0.8),fixtures=wheel2Def)

self.joint1 = world.CreateWheelJoint(bodyA=self.chassis,bodyB=self.wheel1,anchor=(5.5,height-0.8),axis=(0,1),motorSpeed=0.0,maxMotorTorque=5000.0,enableMotor=True,frequencyHz=20.0,dampingRatio=0.5,collideConnected=False)

self.joint2 = world.CreateWheelJoint(bodyA=self.chassis,bodyB=self.wheel2,anchor=(2.5,height-0.8),axis=(0,1),motorSpeed=0.0,maxMotorTorque=5000.0,enableMotor=True,frequencyHz=20.0,dampingRatio=0.5,collideConnected=False)

def Draw(self,window):

angle = 2\*math.pi- self.chassis.angle

x1,y1 = self.wheel1.position

x1 = int(x1\*20)

y1 = int(600-y1\*20)

point3 = (x1+32\*math.sin(angle),y1-32\*math.cos(angle))

point4 = (x1,y1)

x2,y2 = self.wheel2.position

x2 = int(x2\*20)

y2 = int(600-y2\*20)

point2 = (x2+32\*math.sin(angle),y2-32\*math.cos(angle))

point1 = (x2,y2)

pygame.draw.line(window,(0,0,0),point1,point2,5)

pygame.draw.line(window,(0,0,0),point2,point3,5)

pygame.draw.line(window,(0,0,0),point3,point4,5)

pygame.draw.line(window,(0,0,0),point4,point1,5)

pygame.draw.circle(window,(0,0,0),(x1,y1),16,5)

pygame.draw.circle(window,(0,0,0),(x2,y2),16,5)

class bike(vehicle):

def create(self,world,height):

height += 3

chassisDef = b2FixtureDef(shape=b2PolygonShape(box=(1.5,0.5)),density=500,friction=0.3,categoryBits=0x0004,maskBits=0x0002)

chassisDef.filter.groupIndex = -2

self.chassis = world.CreateDynamicBody(position=(4,height),fixtures=chassisDef)

wheel1Def = b2FixtureDef(shape=b2CircleShape(radius=0.8),density=100,categoryBits=0x0004,maskBits=0x0002)

wheel1Def.filter.groupIndex = -2

self.wheel1 = world.CreateDynamicBody(position=(5.5,height-0.5),fixtures=wheel1Def)

wheel2Def = b2FixtureDef(shape=b2CircleShape(radius=0.8),density=100,categoryBits=0x0004,maskBits=0x0002)

wheel2Def.filter.groupIndex = -2

self.wheel2 = world.CreateDynamicBody(position=(2.5,height-0.5),fixtures=wheel2Def)

self.joint1 = world.CreateWheelJoint(bodyA=self.chassis,bodyB=self.wheel1,anchor=(5.5,height-0.5),axis=(0,1),motorSpeed=0.0,maxMotorTorque=5000.0,enableMotor=True,frequencyHz=20.0,dampingRatio=0.5,collideConnected=False)

self.joint2 = world.CreateWheelJoint(bodyA=self.chassis,bodyB=self.wheel2,anchor=(2.5,height-0.5),axis=(0,1),motorSpeed=0.0,maxMotorTorque=5000.0,enableMotor=True,frequencyHz=20.0,dampingRatio=0.5,collideConnected=False)

def Draw(self,window):

angle = 2\*math.pi- self.chassis.angle

x1,y1 = self.wheel1.position

x1 = int(x1\*20)

y1 = int(600-y1\*20)

point3 = (x1+20\*math.sin(angle),y1-20\*math.cos(angle))

point4 = (x1,y1)

x2,y2 = self.wheel2.position

x2 = int(x2\*20)

y2 = int(600-y2\*20)

point2 = (x2+20\*math.sin(angle),y2-20\*math.cos(angle))

point1 = (x2,y2)

pygame.draw.line(window,(0,0,0),point1,point2,5)

pygame.draw.line(window,(0,0,0),point2,point3,5)

pygame.draw.line(window,(0,0,0),point3,point4,5)

pygame.draw.line(window,(0,0,0),point4,point1,5)

pygame.draw.circle(window,(0,0,0),(x1,y1),16,5)

pygame.draw.circle(window,(0,0,0),(x2,y2),16,5)

class truck(vehicle):

def create(self,world,height):

height += 3

chassisDef = b2FixtureDef(shape=b2PolygonShape(box=(2.5,1)),density=500,friction=0.3,categoryBits=0x0004,maskBits=0x0002)

chassisDef.filter.groupIndex = -2

self.chassis = world.CreateDynamicBody(position=(4,height),fixtures=chassisDef)

wheel1Def = b2FixtureDef(shape=b2CircleShape(radius=0.8),density=100,categoryBits=0x0004,maskBits=0x0002)

wheel1Def.filter.groupIndex = -2

self.wheel1 = world.CreateDynamicBody(position=(6.5,height-1),fixtures=wheel1Def)

wheel2Def = b2FixtureDef(shape=b2CircleShape(radius=0.8),density=100,categoryBits=0x0004,maskBits=0x0002)

wheel2Def.filter.groupIndex = -2

self.wheel2 = world.CreateDynamicBody(position=(1.5,height-1),fixtures=wheel2Def)

self.joint1 = world.CreateWheelJoint(bodyA=self.chassis,bodyB=self.wheel1,anchor=(6.5,height-1),axis=(0,1),motorSpeed=0.0,maxMotorTorque=5000.0,enableMotor=True,frequencyHz=20.0,dampingRatio=0.5,collideConnected=False)

self.joint2 = world.CreateWheelJoint(bodyA=self.chassis,bodyB=self.wheel2,anchor=(1.5,height-1),axis=(0,1),motorSpeed=0.0,maxMotorTorque=5000.0,enableMotor=True,frequencyHz=20.0,dampingRatio=0.5,collideConnected=False)

def Draw(self,window):

angle = 2\*math.pi- self.chassis.angle

x1,y1 = self.wheel1.position

x1 = int(x1\*20)

y1 = int(600-y1\*20)

point3 = (x1+40\*math.sin(angle),y1-40\*math.cos(angle))

point4 = (x1,y1)

x2,y2 = self.wheel2.position

x2 = int(x2\*20)

y2 = int(600-y2\*20)

point2 = (x2+40\*math.sin(angle),y2-40\*math.cos(angle))

point1 = (x2,y2)

pygame.draw.line(window,(0,0,0),point1,point2,5)

pygame.draw.line(window,(0,0,0),point2,point3,5)

pygame.draw.line(window,(0,0,0),point3,point4,5)

pygame.draw.line(window,(0,0,0),point4,point1,5)

pygame.draw.circle(window,(0,0,0),(x1,y1),16,5)

pygame.draw.circle(window,(0,0,0),(x2,y2),16,5)

### Build.py

import pygame,sys,Graphics,Classes,BridgeData,Save

from pygame.locals import \*

def loadBridge(bridgeID):

jointList = []

bridgeFile,dif,land = BridgeData.getBridgeFile(bridgeID)

bridgeFile = eval(bridgeFile.strip("'"))

jointNum = len(bridgeFile)

materialStack = []

added = False

for joint in bridgeFile:

jointX,jointY = bridgeFile[joint]['location']

jointList.append({'index':int(joint),'point':(jointX\*20,600 - jointY\*20)})

for joint in bridgeFile:

joint1 = int(joint)

for connectedJoint in bridgeFile[joint]['connectedJoints']:

joint2 = int(connectedJoint['joint'])

for material in materialStack:

if (material.getJoint1() == joint1 and material.getJoint2() == joint2) or (material.getJoint1() == joint2 and material.getJoint2() == joint1):

added = True

if added == False:

if connectedJoint['material'] == 'Steel':

item = Classes.Steel()

elif connectedJoint['material'] == 'Wood':

item = Classes.Wood()

elif connectedJoint['material'] == 'Road':

item = Classes.Road()

elif connectedJoint['material'] == 'Rope':

item = Classes.Rope()

elif connectedJoint['material'] == 'Cable':

item = Classes.Cable()

item.setjoint1(joint1)

item.setjoint2(joint2)

materialStack.append(item)

else:

added = False

for material in materialStack:

for joint in jointList:

if int(material.getJoint1()) == int(joint['index']):

jointX,jointY = joint['point']

material.setCo1(jointX,jointY)

elif int(material.getJoint2()) == int(joint['index']):

jointX,jointY = joint['point']

material.setCo2(jointX,jointY)

return [materialStack,jointList,jointNum,dif,land]

# delete joints with no related materials

def deleteExcessJoint(jointList,materialStack):

toDelete = []

for joint in jointList:

numOfConnectedMaterials = 0

for material in materialStack:

if joint['index'] == material.getJoint1() or joint['index'] == material.getJoint2():

numOfConnectedMaterials += 1

if numOfConnectedMaterials == 0:

toDelete.append(joint)

for joint in toDelete:

jointList.remove(joint)

# returns which joint has been clicked on if any

def jointClick(jointList,dotX,dotY):

for joint in jointList:

jointX,jointY = joint['point']

rect = pygame.Rect(jointX-6,jointY-6,12,12)

if rect.collidepoint((dotX,dotY)) == True:

return joint['index']

return 0

# determines whether a material can be placed in its current location

def allowConnect(materialStack,item,dotX,dotY):

for material in materialStack:

if ((material.getStart() == item.getStart()) and (material.getEnd() == (dotX,dotY))) or ((material.getStart() == (dotX,dotY)) and (material.getEnd() == item.getStart())):

item.errorTrue()

# returns which button has been clicked

def stckbtnclk(stickButtons):

for button in stickButtons:

if stickButtons[button].ifClick():

return button

return ""

# turns on clicked button

def turnOn(btn,stickButtons):

for button in stickButtons:

if button == btn:

stickButtons[btn].turnOn()

else:

stickButtons[button].turnOff()

# determines which button is on

def buttonOn(stickButtons):

for button in stickButtons:

if stickButtons[button].getOn():

return button

return ""

def Main(info,window):

stickButtons = {

'steel':Classes.stickButton(' Steel ',100,550),

'wood':Classes.stickButton(' Wood ',300,550),

'road':Classes.stickButton('Concrete',500,550),

'rope':Classes.stickButton(' Rope ',700,550),

'cable':Classes.stickButton(' Cable ',900,550),

'delete':Classes.stickButton(' Delete ',900,50,(223,12,12)),

'quit':Classes.stickButton(' Quit ',100,50,(223,12,12)),

'save':Classes.stickButton(' Save ',300,50,(34,204,0)),

'undo':Classes.stickButton(' Undo ',700,50,(223,12,12)),

'test':Classes.stickButton(' Test ',500,50)

}

if info['loadBridge'] == False:

materialStack = []

jointList = []

jointNum = 0

else:

materialStack,jointList,jointNum,dif,land = loadBridge(info['bridgeID'])

info['dif'] = dif.strip("'")

info['land'] = land

clickStage = "PlaceMaterial1"

click = False

clock = pygame.time.Clock()

while True:

for event in pygame.event.get():

if event.type == QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

click = True

elif event.type == pygame.MOUSEBUTTONUP:

release = True

Graphics.BackDrop(info['dif'],info['land'],window,True)

Material = buttonOn(stickButtons)

if Material == "delete":

clickStage = "Delete"

elif Material == "quit":

clickStage = "Quit"

click = True

elif Material == "save":

clickStage = "Save"

click = True

elif Material == "undo":

clickStage = "Undo"

click = True

elif Material == "test":

clickStage = "Test"

click = True

for material in materialStack:

material.Draw(window)

for joint in jointList:

Graphics.drawJoint(window,joint['point'])

if clickStage == "PlaceMaterial1":

for button in stickButtons:

stickButtons[button].create(window)

elif clickStage == "PlaceMaterial2":

item.updateLocation(dotX,dotY)

item.checkPlacement()

allowConnect(materialStack,item,dotX,dotY)

item.Draw(window)

if Material != "":

if Material == "delete":

dotX,dotY = Graphics.drawDot(window,info['dif'],info['land'],(233,12,12))

else:

dotX,dotY = Graphics.drawDot(window,info['dif'],info['land'],(255,255,255))

if click:

if clickStage == "PlaceMaterial1":

if Material != "" and stckbtnclk(stickButtons) == "":

if Material == "steel":

item = Classes.Steel()

elif Material == "wood":

item = Classes.Wood()

elif Material == "road":

item = Classes.Road()

elif Material == "rope":

item = Classes.Rope()

elif Material == "cable":

item = Classes.Cable()

item.setCo1(dotX,dotY)

clickStage = "PlaceMaterial2"

jointPoints = []

for joint in jointList:

jointPoints.append(joint['point'])

if (dotX,dotY) not in jointPoints:

jointNum += 1

jointList.append({'index':jointNum,'point':(dotX,dotY)})

for joint in jointList:

if joint['point'] == (dotX,dotY):

item.setjoint1(joint['index'])

else:

turnOn(stckbtnclk(stickButtons),stickButtons)

elif clickStage == "PlaceMaterial2":

if item.getError() == False:

jointPoints = []

for joint in jointList:

jointPoints.append(joint['point'])

if (dotX,dotY) not in jointPoints:

jointNum += 1

jointList.append({'index':jointNum,'point':(dotX,dotY)})

for joint in jointList:

if joint['point'] == (dotX,dotY):

item.setjoint2(joint['index'])

materialStack.append(item)

clickStage = "PlaceMaterial1"

elif clickStage == "Delete":

jointClicked = jointClick(jointList,dotX,dotY)

jointCount = 0

cap = False

toDelete = []

if jointClicked != 0:

for joint in jointList:

if joint['index'] == jointClicked:

cap = True

if cap == False:

jointCount+=1

del jointList[jointCount]

for material in materialStack:

if material.getJoint1() == jointClicked or material.getJoint2() == jointClicked:

toDelete.append(material)

for material in toDelete:

materialStack.remove(material)

deleteExcessJoint(jointList,materialStack)

clickStage = "PlaceMaterial1"

stickButtons['delete'].turnOff()

elif clickStage == "Quit":

info['build'] = False

return ["sec",info]

elif clickStage == "Save":

if len(materialStack) != 0:

Save.Main(window,jointList,materialStack,info)

clickStage = "PlaceMaterial1"

stickButtons['save'].turnOff()

elif clickStage == "Undo":

if len(materialStack) != 0:

del materialStack[-1]

clickStage = "PlaceMaterial1"

stickButtons['undo'].turnOff()

deleteExcessJoint(jointList,materialStack)

elif clickStage == "Test":

if len(materialStack) != 0:

Save.Main(window,jointList,materialStack,info)

info['build'] = False

return ["chooseVehicle",info]

clickStage = "PlaceMaterial1"

stickButtons['test'].turnOff()

click = False

clock.tick()

fps = clock.get\_fps()

#print(fps)

pygame.display.update()

### Test.py

from Box2D import \*

import pygame,sys,math,Graphics,Classes,Build

def createJoints(joint,world,groundBody1,groundBody2):

for materialA in joint['materials']:

for materialB in joint['materials']:

if materialA != materialB:

if materialA.getMaterial() == "Rope" or materialA.getMaterial()== "Cable":

if joint['index'] == materialA.getJoint1():

bodyA = materialA.getBody()[0]

elif joint['index'] == materialA.getJoint2():

bodyA = materialA.getBody()[-1]

else:

bodyA = materialA.getBody()

if materialB.getMaterial() == "Rope" or materialB.getMaterial()== "Cable":

if joint['index'] == materialB.getJoint1():

bodyB = materialB.getBody()[0]

elif joint['index'] == materialB.getJoint2():

bodyB = materialB.getBody()[-1]

else:

bodyB = materialB.getBody()

Joint = world.CreateRevoluteJoint(bodyA=bodyA,bodyB=bodyB,anchor=joint['point'],collideConnected=False)

if joint['index'] == materialB.getJoint1():

materialB.addB2joints1({"reference":Joint,"maxForce":materialA.getMaxForce()})

elif joint['index'] == materialB.getJoint2():

materialB.addB2joints2({"reference":Joint,"maxForce":materialA.getMaxForce()})

if joint['onGround1']:

bodyA = groundBody1

elif joint['onGround2']:

bodyA = groundBody2

if joint['onGround1'] or joint['onGround2']:

for material in joint['materials']:

if material.getMaterial() == "Rope" or material.getMaterial()== "Cable":

if joint['index'] == material.getJoint1():

bodyB = material.getBody()[0]

elif joint['index'] == material.getJoint2():

bodyB = material.getBody()[-1]

else:

bodyB = material.getBody()

Joint = world.CreateRevoluteJoint(bodyA=bodyA,bodyB=bodyB,anchor=joint['point'],collideConnected=False)

if joint['index'] == material.getJoint1():

material.addB2joints1({"reference":Joint,"maxForce":material.getMaxForce()})

elif joint['index'] == material.getJoint2():

material.addB2joints2({"reference":Joint,"maxForce":material.getMaxForce()})

def loadBridge(bridgeID,world):

materialStack,jointList,jointNum,dif,land = Build.loadBridge(bridgeID)

Dirt1Y,Dirt1width,Dirt1height,Dirt2X,Dirt2Y,Dirt2width,Dirt2height = Graphics.dirtSize(dif,land)

Dirt1Y = 30 - Dirt1Y/20

Dirt1width = Dirt1width/20

Dirt1height = Dirt1height/20

Dirt2X = Dirt2X/20

Dirt2Y = 30 - Dirt2Y/20

Dirt2width = Dirt2width/20

Dirt2height = Dirt2height/20

for joint in jointList:

convx,convy = joint['point']

joint['point'] = (convx/20,30-convy/20)

GB1fix = b2FixtureDef(shape=b2PolygonShape(box=(Dirt1width/2,Dirt1height/2)),friction=0.2,categoryBits=0x0002,maskBits=0x0004)

groundBody1 = world.CreateStaticBody(position=(Dirt1width/2,Dirt1height/2),fixtures = GB1fix)

GB2fix = b2FixtureDef(shape=b2PolygonShape(box=(Dirt2width/2,Dirt2height/2)),friction=0.2,categoryBits=0x0002,maskBits=0x0004)

groundBody2 = world.CreateStaticBody(position=(50-Dirt2width/2,Dirt2height/2),fixtures = GB2fix)

wallfix = b2FixtureDef(shape=b2PolygonShape(box=(1,15)),friction=0.2,categoryBits=0x0002,maskBits=0x0004)

wall1 = world.CreateStaticBody(position=(-1,15),fixtures = wallfix)

wall2 = world.CreateStaticBody(position=(51,15),fixtures = wallfix)

for material in materialStack:

material.createBody(world)

for joint in jointList:

materialAboutJoint = []

for material in materialStack:

if joint['index'] == material.getJoint1() or joint['index'] == material.getJoint2():

materialAboutJoint.append(material)

joint['materials'] = materialAboutJoint

jointx,jointy = joint['point']

if (jointx == Dirt1width and jointy <= Dirt1Y) or (jointx <= Dirt1width and jointy == Dirt1Y):

joint['onGround1'] = True

joint['onGround2'] = False

elif (jointx == Dirt2X and jointy <= Dirt2Y) or (jointx >= Dirt2X and jointy == Dirt2Y):

joint['onGround1'] = False

joint['onGround2'] = True

else:

joint['onGround1'] = False

joint['onGround2'] = False

createJoints(joint,world,groundBody1,groundBody2)

return (materialStack,jointList,dif,land)

def Main(info,window):

world = b2World()

materialStack,jointList,dif,land = loadBridge(info['bridgeID'],world)

clock = pygame.time.Clock()

timeStep = 1.0/25

vel\_iters,pos\_iters = 7,7

frameCount = 0

font = pygame.font.SysFont(None,90)

text1 = font.render('Loading', True,(255,255,255),None)

text1Rect = text1.get\_rect()

text1Rect.centerx = window.get\_rect().centerx

text1Rect.centery = window.get\_rect().centery

text2 = font.render('Loading', True,(0,0,0),None)

text2Rect = text2.get\_rect()

text2Rect.centerx = window.get\_rect().centerx - 5

text2Rect.centery = window.get\_rect().centery - 5

buttons = {

'quit':Classes.Button(' Quit ',150,50),

'edit':Classes.Button(' Edit ',400,50)

}

if info['vehicle'] == "car":

vehicle = Classes.car()

elif info['vehicle'] == "bike":

vehicle = Classes.bike()

elif info['vehicle'] == "truck":

vehicle = Classes.truck()

if land == 1:

height = 10

elif land == 2:

height = 13

elif land == 3:

height = 8

vehicle.create(world,height)

forward,back,click = False,False,False

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

click = True

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_d:

if back == False:

forward = True

elif event.key == pygame.K\_a:

if forward == False:

back = True

elif event.type == pygame.KEYUP:

if event.key == pygame.K\_d:

forward = False

elif event.key == pygame.K\_a:

back = False

if frameCount%5 == 0 and frameCount > 100:

for material in materialStack:

material.ifBreak(world,timeStep)

Graphics.BackDrop(dif,land,window)

for material in materialStack:

material.testDraw(window)

vehicle.Draw(window)

if frameCount > 100:

if forward:

vehicle.forward()

elif back:

vehicle.back()

elif (forward == False and back != True) or (back == False and forward != True):

vehicle.stop()

else:

window.blit(text2,text2Rect)

window.blit(text1,text1Rect)

frameCount += 1

buttons['quit'].create(window)

buttons['edit'].create(window)

if click:

if buttons['quit'].ifClick():

info['test'] = False

return ["sec",info]

elif buttons['edit'].ifClick():

info['test'] = False

info['loadBridge'] = True

info['build'] = True

return ["build",info]

world.Step(timeStep,vel\_iters,pos\_iters)

world.ClearForces()

#clock.tick()

#print(clock.get\_fps())

pygame.display.update()

pygame.time.wait(5)

### BridgeData.py

import pymysql

def findUsername(username):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = "SELECT \* FROM User WHERE Username = '%s';" %(username)

results = ""

try:

cursor.execute(command)

if len(cursor.fetchall()) > 0:

results = "Taken"

except:

results = "Error"

db.close()

return results

def findEmail(email):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = "SELECT \* FROM User WHERE Email = '%s';" %(email)

results = ""

try:

cursor.execute(command)

if len(cursor.fetchall()) > 0:

results = "Taken"

except:

results = "Error"

db.close()

return results

def addUser(first,sec,email,username,password):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = ("INSERT INTO User (Username,First\_Name,Surname,Email,Password) VALUES ('%s','%s','%s','%s','%s');"%(username,first,sec,email,password))

try:

cursor.execute(command)

db.commit()

error = ""

except:

db.rollback()

error = "Error connecting to database"

db.close()

return error

def findUser(username,password):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = ("SELECT \* FROM User WHERE Username = '%s' and Password = '%s';"%(username,password))

results = ""

try:

cursor.execute(command)

if len(cursor.fetchall()) == 0:

results = "Not"

except:

results = "Error"

db.close()

return results

def getUser\_ID(username):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = "SELECT User\_ID FROM User WHERE Username = '%s';" %(username)

ID = ""

try:

cursor.execute(command)

result = cursor.fetchall()

ID = result[0][0]

except:

ID = ""

db.close()

return ID

def findBridge(name,User\_ID):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = "SELECT \* FROM Bridges WHERE User\_ID = '%s' AND Bridge\_Name = '%s';" %(User\_ID,name)

results = ""

try:

cursor.execute(command)

if len(cursor.fetchall()) > 0:

results = "Taken"

except:

results = "Error"

db.close()

return results

def addBridge(name,adjacencyList,info):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = ('INSERT INTO Bridges (User\_ID,Bridge\_Name,Date\_Last\_Edit,BridgeFile,Difficulty,Land\_Type) VALUES ("%s","%s",CURDATE(),"%s","%s","%s");'%(info['User\_ID'],name,adjacencyList,info['dif'],info['land']))

try:

cursor.execute(command)

db.commit()

error = ""

except pymysql.Error as e:

# print(e)

db.rollback()

error = "Error"

db.close()

return error

def getBridges(User\_ID):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = "SELECT Bridge\_ID,Bridge\_Name,DAYOFMONTH(Date\_Last\_Edit),MONTH(Date\_Last\_Edit),YEAR(Date\_Last\_Edit) FROM Bridges WHERE User\_ID = '%s';" %(User\_ID)

results = ""

try:

cursor.execute(command)

results = cursor.fetchall()

except:

results = "Error"

db.close()

return results

def deleteBridge(bridgeID):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = ("DELETE FROM Bridges WHERE Bridge\_ID = '%s'"%(bridgeID))

try:

cursor.execute(command)

db.commit()

except:

db.rollback()

db.close()

def getBridgeFile(bridgeID):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = ("SELECT BridgeFile,Difficulty,Land\_Type FROM Bridges WHERE Bridge\_ID = '%s'"%(bridgeID))

try:

cursor.execute(command)

results = cursor.fetchall()

except:

pass

db.close()

return results[0]

def updateBridge(bridgeID,adjacencyList):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = ('UPDATE Bridges SET BridgeFile = "%s", Date\_Last\_Edit = CURDATE() WHERE Bridge\_ID = "%s";'%(adjacencyList,bridgeID))

try:

cursor.execute(command)

db.commit()

except pymysql.Error as e:

#print(e)

db.rollback()

def getBridgeID(name,ID):

db = pymysql.connect("localhost","Kiran","!winter@2018","BridgeBuilder")

cursor = db.cursor()

command = ("SELECT Bridge\_ID FROM Bridges WHERE (User\_ID = '%s' AND Bridge\_Name = '%s')"%(ID,name))

try:

cursor.execute(command)

results = cursor.fetchall()

except pymysql.Error as e:

# print(e)

pass

return results[0][0]

### Save.py

import Classes

import pygame,sys

import BridgeData

import os

def checkSaveName(inputboxes,info,adjacencyList):

checkList = [" ", ";", "=", "'", '"']

error = []

emptyCount = 0

bridgeID = ""

for box in inputboxes:

for char in checkList:

if (inputboxes[box].getCap() == "") or (char in inputboxes[box].getCap()):

emptyCount += 1

if emptyCount > 0:

text = "All fields must be entered and cannot contain ;, =, ' or \""

error.append(text)

if error == []:

name = inputboxes['name'].getCap()

nameState = BridgeData.findBridge(name,info['User\_ID'])

if nameState == "Taken":

error.append("You have used this name already")

elif nameState == "Error":

error.append("Error connecting to database")

if error == []:

fileName = (name+"\_"+str(info['User\_ID']))

with open(fileName,'w') as File:

File.write(str(adjacencyList))

with open(fileName,'r') as File:

openedFile = File.read()

connect = BridgeData.addBridge(name,openedFile,info)

os.remove(fileName)

if connect != "":

error.append("Error connecting to database")

else:

bridgeID = BridgeData.getBridgeID(name,info['User\_ID'])

return [error,bridgeID]

# produces a message on screen

def PopUpMessage(text,window):

center = (window.get\_rect().centerx, window.get\_rect().centery)

y = 50\*(len(text)+1)

message = Classes.TextBox(text,y,950,center,(233,12,12))

message.create(window)

# creates buttons and input box on screen

def createSave(buttons,inputboxes,window,error):

buttons['save'].create(window)

buttons['cancel'].create(window)

if error == True:

buttons['ok'].create(window)

height = 100

width = 800

x = window.get\_rect().centerx

y = 280

center = (x,y)

text = Classes.TextBox("",height,width,center)

text.create(window)

inputboxes['name'].create(window)

def findConnections(index,materials):

connectedTo = []

for material in materials:

if index == material.getJoint1():

connectedTo.append({'joint':material.getJoint2(),'material':material.getMaterial()})

elif index == material.getJoint2():

connectedTo.append({'joint':material.getJoint1(),'material':material.getMaterial()})

return connectedTo

def Main(window,jointList,materialStack,info):

adjacencyList = {}

for joint in jointList:

index = joint['index']

x,y = joint['point']

x /= 20

y = 30 - y/20

location = (int(x),int(y))

connectedJoints = findConnections(index,materialStack)

adjacencyList[str(index)] = {'location':location, 'connectedJoints':connectedJoints}

buttons = {

'save':Classes.Button(' Save ',650,500),

'cancel':Classes.Button(' Cancel ',350,500),

'ok':Classes.Button(' OK ',500,500,(233,12,12))

}

inputboxes = {

'name':Classes.InputBox("Bridge Name:",365,250,470,60)

}

click = False

Type = False

error = False

if info['loadBridge'] == False:

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

click = True

elif event.type == pygame.KEYDOWN:

Type = True

if event.key == pygame.K\_BACKSPACE:

char = 'back'

else:

char = event.unicode

createSave(buttons,inputboxes,window,error)

if error == True:

PopUpMessage(sentences,window)

if click == True:

if error == False:

inputboxes['name'].deActive()

if buttons['save'].ifClick():

sentences,bridgeID = checkSaveName(inputboxes,info,adjacencyList)

info['bridgeID'] = bridgeID

if sentences != []:

error = True

else:

error = False

info['loadBridge'] = True

return

elif buttons['cancel'].ifClick():

return

elif inputboxes['name'].ifClick():

inputboxes['name'].makeActive()

else:

if buttons['ok'].ifClick():

error = False

return

click = False

if Type == True:

if inputboxes['name'].getActive():

if char == 'back':

inputboxes['name'].Back()

elif (char != 'back') and (inputboxes['name'].getCapLength() < 20):

inputboxes['name'].Add(char)

Type = False

pygame.display.update()

else:

fileName = ("Temp")

with open(fileName,'w') as File:

File.write(str(adjacencyList))

with open(fileName,'r') as File:

openedFile = File.read()

BridgeData.updateBridge(info['bridgeID'],adjacencyList)

os.remove(fileName)

### Graphics.py

import pygame

import math

White = (255,255,255)

Black = (0,0,0)

DirtBrown = (97,63,16)

GrassGreen = (44,176,55)

CloudWhite = (250,242,248)

oceanBlue = (79,66,181)

skyBlue = (135,206,235)

DarkBrown = (51,0,0)

LightBrown = (204,102,0)

# creates octogon shape on screen

def generateOct(window,x,y):

poly1 = ((x-110,y+105),(x-105,y+110),(x+105,y+110),(x+110,y+105),(x+110,y-105),(x+105,y-110),(x-105,y-110),(x-110,y-105))

poly2 = ((x-115,y+100),(x-110,y+105),(x+100,y+105),(x+105,y+100),(x+105,y-110),(x+100,y-115),(x-110,y-115),(x-115,y-110))

poly3 = ((x-95,y+90),(x-90,y+95),(x+90,y+95),(x+95,y+90),(x+95,y-90),(x+90,y-95),(x-90,y-95),(x-95,y-90))

pygame.draw.polygon(window,DarkBrown,poly2)

pygame.draw.polygon(window,LightBrown,poly1)

pygame.draw.polygon(window,skyBlue,poly3)

# creates the images of the different types of lands

def LandImage(window):

centerY = 300

var = 20

for centerX in range(250,1000,250):

generateOct(window,centerX,centerY)

poly1 = ((centerX-90,centerY+95),(centerX-95,centerY+90),(centerX-95,centerY-var),(centerX-35,centerY-var),(centerX-35,centerY+95))

poly2 = ((centerX+90,centerY+95),(centerX+95,centerY+90),(centerX+95,centerY+var),(centerX+35,centerY+var),(centerX+35,centerY+95))

pygame.draw.polygon(window,DirtBrown,poly1)

pygame.draw.polygon(window,DirtBrown,poly2)

pygame.draw.rect(window,GrassGreen,(centerX-95,centerY-var,61,10))

pygame.draw.rect(window,GrassGreen,(centerX+35,centerY+var,61,10))

var -= 20

def DifImage(window,op):

if op == 1:

yvar = 0

elif op == 2:

yvar = -20

elif op == 3:

yvar = 20

yloc = 300

xvar = 15

width = 81

for xloc in range(250,1000,250):

generateOct(window,xloc,yloc)

poly1 = ((xloc-90,yloc+95),(xloc-95,yloc+90),(xloc-95,yloc+yvar),(xloc-xvar,yloc+yvar),(xloc-xvar,yloc+95))

poly2 = ((xloc+90,yloc+95),(xloc+95,yloc+90),(xloc+95,yloc-yvar),(xloc+xvar,yloc-yvar),(xloc+xvar,yloc+95))

pygame.draw.polygon(window,DirtBrown,poly1)

pygame.draw.polygon(window,DirtBrown,poly2)

pygame.draw.rect(window,GrassGreen,(xloc-95,yloc+yvar,width,10))

pygame.draw.rect(window,GrassGreen,(xloc+xvar,yloc-yvar,width,10))

xvar += 20

width -= 20

# creates the images of the vehicle menu

def vehicleImage(window):

for xloc in range(250,1000,250):

generateOct(window,xloc,300)

poly = ((xloc-90,395),(xloc-95,390),(xloc-95,330),(xloc+95,330),(xloc+95,390),(xloc+90,395))

pygame.draw.polygon(window,DirtBrown,poly)

pygame.draw.rect(window,GrassGreen,(xloc-95,330,190,11))

pygame.draw.line(window,Black,(470,285),(530,285),5)

pygame.draw.line(window,Black,(470,285),(470,315),5)

pygame.draw.line(window,Black,(530,285),(530,315),5)

pygame.draw.line(window,Black,(470,315),(530,315),5)

pygame.draw.circle(window,Black,(470,315),15,5)

pygame.draw.circle(window,Black,(530,315),15,5)

pygame.draw.line(window,Black,(220,295),(280,295),5)

pygame.draw.line(window,Black,(220,295),(220,315),5)

pygame.draw.line(window,Black,(280,295),(280,315),5)

pygame.draw.line(window,Black,(220,315),(280,315),5)

pygame.draw.circle(window,Black,(220,315),15,5)

pygame.draw.circle(window,Black,(280,315),15,5)

pygame.draw.line(window,Black,(710,275),(790,275),5)

pygame.draw.line(window,Black,(710,275),(710,315),5)

pygame.draw.line(window,Black,(790,275),(790,315),5)

pygame.draw.line(window,Black,(710,315),(790,315),5)

pygame.draw.circle(window,Black,(710,315),15,5)

pygame.draw.circle(window,Black,(790,315),15,5)

# defines the size of the ground

def dirtSize(dif,land):

if dif == "easy":

Dirt1width = 340

Dirt2width = 340

Dirt2X = 660

elif dif == "normal":

Dirt1width = 240

Dirt2width = 240

Dirt2X = 760

elif dif == "hard":

Dirt1width = 140

Dirt2width = 140

Dirt2X = 860

if land == 1:

Dirt1height = 200

Dirt2height = 200

Dirt1Y = 400

Dirt2Y = 400

elif land == 2:

Dirt1height = math.ceil((200 + ((Dirt2X-Dirt1width)\*(17/100))/2)/20)\*20

Dirt2height = math.ceil((200 - ((Dirt2X-Dirt1width)\*(17/100))/2)/20)\*20

Dirt1Y = 600 - Dirt1height

Dirt2Y = 600 - Dirt2height

elif land == 3:

Dirt1height = math.ceil((200 - ((Dirt2X-Dirt1width)\*(17/100))/2)/20)\*20

Dirt2height = math.ceil((200 + ((Dirt2X-Dirt1width)\*(17/100))/2)/20)\*20

Dirt1Y = 600 - Dirt1height

Dirt2Y = 600 - Dirt2height

return Dirt1Y,Dirt1width,Dirt1height,Dirt2X,Dirt2Y,Dirt2width,Dirt2height

# creates the background for the menus, build and test

def BackDrop(dif,land,window,build = False):

window.fill((135,206,235))

if dif != "" and land != "":

Dirt1Y,Dirt1width,Dirt1height,Dirt2X,Dirt2Y,Dirt2width,Dirt2height = dirtSize(dif,land)

dirt1 = {'shape':pygame.Rect(0,Dirt1Y,Dirt1width,Dirt1height),'colour':DirtBrown}

dirt2 = {'shape':pygame.Rect(Dirt2X,Dirt2Y,Dirt2width,Dirt2height),'colour':DirtBrown}

grass1 = {'shape':pygame.Rect(0,Dirt1Y,Dirt1width,10),'colour':GrassGreen}

grass2 = {'shape':pygame.Rect(Dirt2X,Dirt2Y,Dirt2width,10),'colour':GrassGreen}

else:

dirt1 = {'shape':pygame.Rect(0,410,150,190),'colour':DirtBrown}

dirt2 = {'shape':pygame.Rect(850,410,150,190),'colour':DirtBrown}

grass1 = {'shape':pygame.Rect(0,400,150,10),'colour':GrassGreen}

grass2 = {'shape':pygame.Rect(850,400,150,10),'colour':GrassGreen}

font = pygame.font.SysFont(None,90)

text1 = font.render('Bridge Builder', True,White,None)

text1Rect = text1.get\_rect()

text1Rect.centerx = window.get\_rect().centerx

text1Rect.centery = 75

text2 = font.render('Bridge Builder', True,Black,None)

text2Rect = text2.get\_rect()

text2Rect.centerx = window.get\_rect().centerx - 5

text2Rect.centery = 70

window.blit(text2,text2Rect)

window.blit(text1,text1Rect)

cloudCircle11 = {'colour':CloudWhite,'pos':(100,150),'radius':50}

cloudCircle12 = {'colour':CloudWhite,'pos':(200,150),'radius':50}

cloudCircle13 = {'colour':CloudWhite,'pos':(150,125),'radius':45}

cloudCircle21 = {'colour':CloudWhite,'pos':(800,250),'radius':50}

cloudCircle22 = {'colour':CloudWhite,'pos':(900,250),'radius':50}

cloudCircle23 = {'colour':CloudWhite,'pos':(850,225),'radius':45}

cloudRect1 = {'shape':pygame.Rect(100,150,100,50),'colour':CloudWhite}

cloudRect2 = {'shape':pygame.Rect(800,250,100,50),'colour':CloudWhite}

circles = [cloudCircle11,cloudCircle12,cloudCircle13,cloudCircle21,cloudCircle22,cloudCircle23]

ocean = True

for xloc in range(0,1000,50):

if ocean == True:

colour = oceanBlue

ocean = False

else:

colour = skyBlue

ocean = True

circles.append({'colour':colour,'pos':(xloc,550),'radius':25})

pygame.draw.rect(window,oceanBlue,pygame.Rect(0,550,1000,50))

boxes = [cloudRect1,cloudRect2,dirt1,dirt2,grass1,grass2]

for circle in circles:

pygame.draw.circle(window,circle['colour'],circle['pos'],circle['radius'],0)

for box in boxes:

pygame.draw.rect(window,box['colour'],box['shape'])

if dif != "" and land != "" and build == True:

for xloc in range(0,1020,20):

pygame.draw.line(window,Black,(xloc,0),(xloc,600),1)

for yloc in range(0,620,20):

pygame.draw.line(window,Black,(0,yloc),(1000,yloc),1)

# determines the loaction of the dot

def dotLoc(dif,land):

mouseLocX,mouseLocY = pygame.mouse.get\_pos()

dotLocMeterX = round(mouseLocX/20)

dotLocMeterY = round(mouseLocY/20)

Dirt1Y,Dirt1width,Dirt1height,Dirt2X,Dirt2Y,Dirt2width,Dirt2height = dirtSize(dif,land)

xBound1 = math.floor(Dirt1width/20)

xBound2 = math.ceil(Dirt2X/20)

yBound1 = math.ceil(Dirt1Y/20)

yBound2 = math.ceil(Dirt2Y/20)

if dotLocMeterX < xBound1 and dotLocMeterY > yBound1:

if (xBound1-dotLocMeterX) < (dotLocMeterY-yBound1):

dotLocMeterX = xBound1

elif (xBound1-dotLocMeterX) > (dotLocMeterY-yBound1):

dotLocMeterY = yBound1

else:

dotLocMeterY = yBound1

elif dotLocMeterX > xBound2 and dotLocMeterY > yBound2:

if (dotLocMeterX-xBound2) < (dotLocMeterY-yBound2):

dotLocMeterX = xBound2

elif (dotLocMeterX-xBound2) > (dotLocMeterY-yBound2):

dotLocMeterY = yBound2

else:

dotLocMeterY = yBound2

dotLocPixelX = dotLocMeterX\*20

dotLocPixelY = dotLocMeterY\*20

return dotLocPixelX,dotLocPixelY

# draws dot

def drawDot(window,dif,land,colour):

x,y = dotLoc(dif,land)

pygame.draw.circle(window,Black,(x,y),6,0)

pygame.draw.circle(window,colour,(x,y),4,0)

return x,y

# draws joint

def drawJoint(window,point):

x,y = point

pygame.draw.rect(window,Black,(x-6,y-6,12,12))

pygame.draw.rect(window,LightBrown,(x-4,y-4,8,8))