BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:		**Surname**	**Given Names**	**Student ID Number**
	1.	**Kurli**	**Georgius**	**2101704571**

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1AC**	**Name of Lecturer(s)**	**:** 1. IDA BAGUS KERTHYAYANA 2. TRI ASIH BUDIONO
Major	**: CS**		
Title of Assignment (if any)	: Tree Cutting Simulation		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 7-11-2017**	**Submission Date**	**:**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

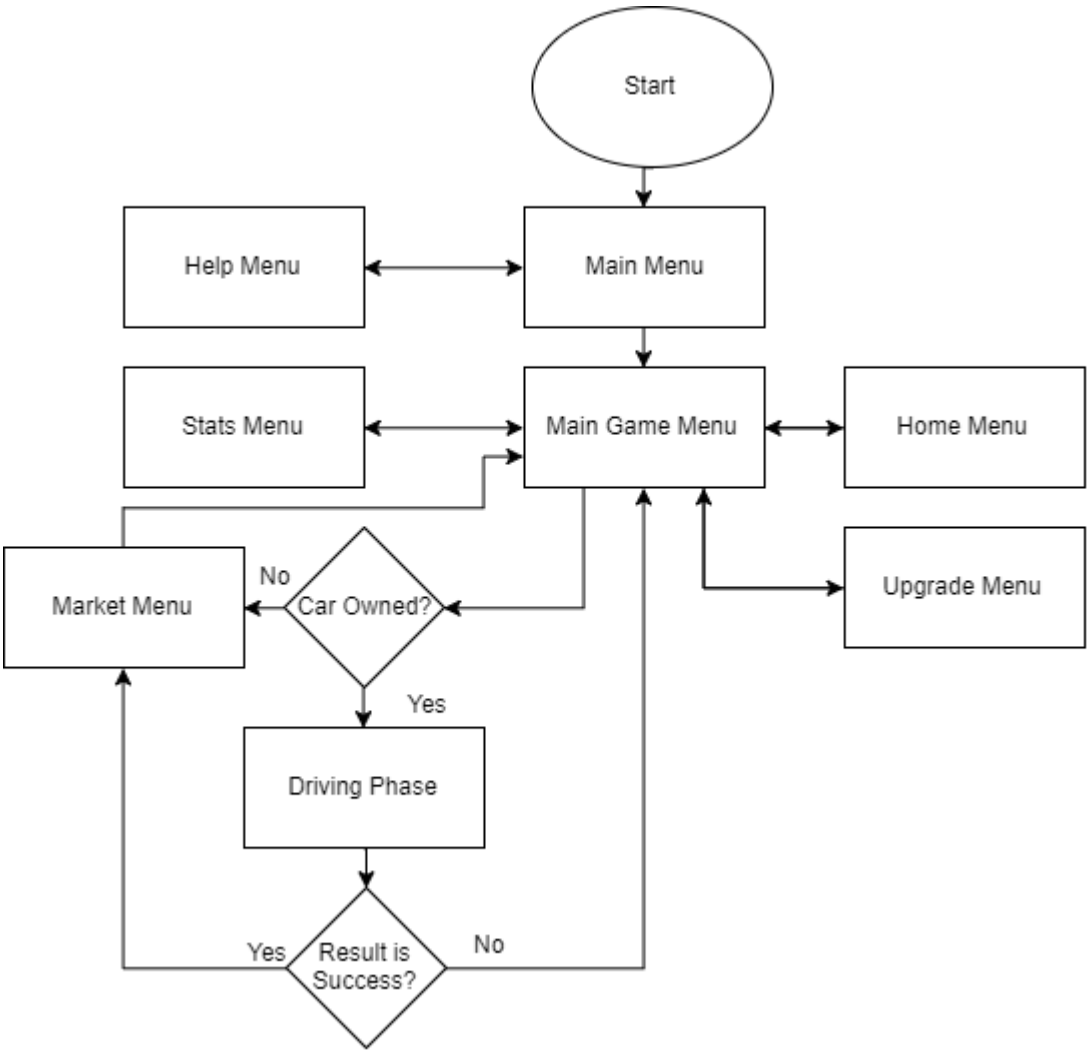
1. Georgius Easton Kuri

Tree Cutting Simulation Game

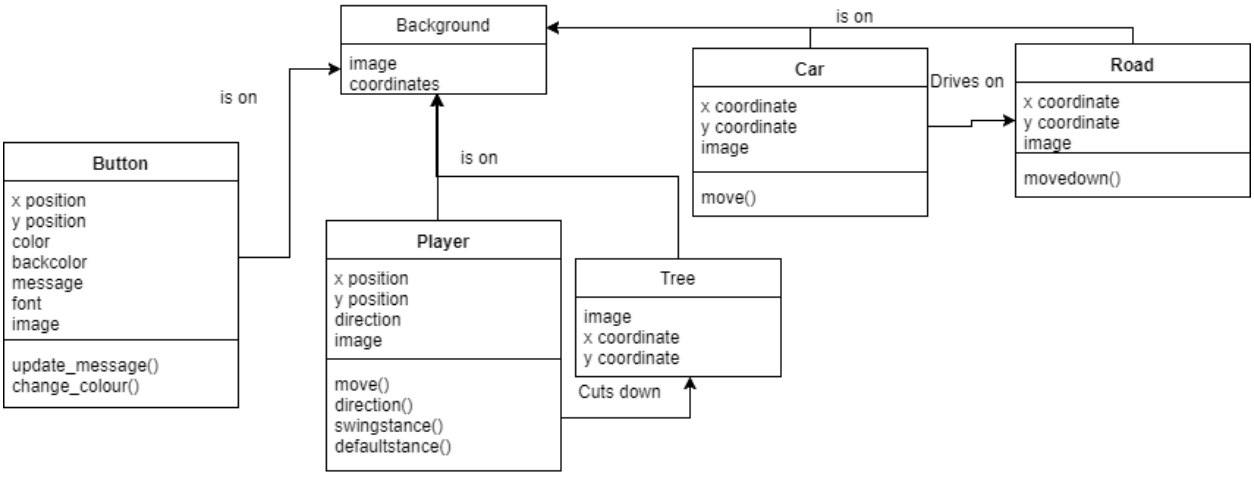
Introduction

This program is an arcade-like game that focuses on earning money from selling logs. The purpose of this program is to entertain people who feels bored.

Flowchart



UML Diagram


Background
image
coordinates
is on
Car
x coordinate
y coordinate
image
move()
Drives on
Road
x coordinate
y coordinate
image
movedown()
is on
Button
x position
y position
color
backcolor
message
font
image
update_message()
change_colour()
is on
Player
x position
y position
direction
image
move()
direction()
swingstance()
defaultstance()
Tree
image
x coordinate
y coordinate
Cuts down

Explanation of Classes in Game_Classes.py:

Button Class

```
#Class to make buttons
class Button(Sprite):
    def __init__(self, message, x,y,fontsize = 42, backcolor = None, color = (0,0,0)):
        Sprite.__init__(self)
        self.color = color
        self.backcolor = backcolor
        self.x = x
        self.y = y
        self.message = message
        self.font = pygame.font.Font(None, fontsize)
        self.image = self.font.render(message,1, color, backcolor)
        self.rect = self.image.get_rect()
        self.rect.center = (x,y)

    #used to update what the buttons display
    def update_message(self, message):
        self.image = self.font.render(message,1,self.color, self.backcolor)
        self.message = message
        self.rect = self.image.get_rect()
        self.rect.center = (self.x,self.y)

    def change_colour(self,color):
        self.color = color
        self.image = self.font.render(self.message,1,color, self.backcolor)
```

This class is used to make buttons that require text to be printed.

The class has default values such as font size 42, no backcolor and white font color. The class has a function “update_message” that updates the object with a new message. The function “change_colour” is used to change the text colour.

Example:

```
#creating buttons
start_button = cl.Button("Press to Start", 540,360)
help_button = cl.Button("How to Play", 540,440, backcolor = cl.light_green)
```

```
#updates money and logs
money_button.update_message("Money: %d"%money)
log_button.update_message("Log: %d"%log)
```

BackGround Class

```
#Class to make background
class BackGround(Sprite):
    def __init__(self, imagefile):
        Sprite.__init__(self)
        self.image = image.load(imagefile)
        self.rect = self.image.get_rect()
        self.rect.center = (540,360)
```

This class is used to create a background sprite at the center of the screen.

Example:

```
main_game_background = cl.BackGround("MainGameBackground_V2.png")
main_game_backgroundup = cl.BackGround("MainGameBackgroundup_V1.png")
```

Player Class

```
#Class to show player
class Player(Sprite):
    def __init__(self):
        Sprite.__init__(self)
        self.image = image.load("Axeman 1(right).png")
        self.rect = self.image.get_rect()
        self.x = 540
        self.y = 360
        self.rect.left = self.x
        self.rect.top = self.y
        self.dir = "right"
```

This class is used to create the player.

The player starts at the center of the screen facing the right.

Move function in Player class

```
#used to move image
def move(self,x,y):
    self.x +=x
    self.y +=y
    self.rect.left = self.x
    self.rect.top = self.y
```

This function is used to move the player image up, down, left or right.

Example:

```
#Checks if movement is true and direction
if moveleft == True:
    if player.x >= 8:
        player.move(-5,0)
```

Direction function in Player class

```
#set image direction
def direction(self,dir):
    self.dir = dir
    if dir == "left":
        self.image = image.load("Axeman 1(left).png")
        self.rect = self.image.get_rect()

    if dir == "right":
        self.image = image.load("Axeman 1(right).png")
        self.rect = self.image.get_rect()
    self.rect.left = self.x
    self.rect.top = self.y
```

This function is used to check the player's direction when moving and to change where the player is facing.

Example:

```
#checks if button is being pressed
elif ev.type == KEYDOWN:
    if ev.key == K_LEFT:
        moveleft = True
        player.direction("left")
```

Swingstance function in Player class

```
#sets the player into swing mode
def swingstance(self):
    if self.dir == "left":
        self.image = image.load("Axeman 1swingleft.png")
        self.rect = self.image.get_rect()

    if self.dir == "right":
        self.image = image.load("Axeman 1(Swinging).png")
        self.rect = self.image.get_rect()

    self.rect.left = self.x
    self.rect.top = self.y
```

This function is used to set the player into swing mode by changing the character image. It also takes player direction into account.

Example:

```
#checks if player is swinging
if swingmode == True:
    player.swingstance()
```

Defaultstance function in player class

```
def defaultstance(self):
    if self.dir == "left":
        self.image = image.load("Axeman 1(left).png")
        self.rect = self.image.get_rect()

    if self.dir == "right":
        self.image = image.load("Axeman 1(right).png")
        self.rect = self.image.get_rect()

    self.rect.left = self.x
    self.rect.top = self.y
```

This function is used to set the player back into idle mode.

Example:

```
if swingmode == False:
    player.defaultstance()
```

Tree Class

```
class Tree(Sprite):
    def __init__(self):
        Sprite.__init__(self)
        self.image = image.load("Tree1(HitBox).png")
        self.rect = self.image.get_rect()
        self.image = image.load("Tree1.png")
        self.x = random.randint(0,1030)
        self.y = random.randint(102,532)
        self.rect.left = self.x
        self.rect.top = self.y
```

This class is used to create the tree sprites. It spawns the tree in a range of coordinates using randomint.

Example:

```
#checks if it is 3 seconds and spawns a tree
if pygame.time.get_ticks() - tree_timer >= 3000:
    temptree = cl.Tree()
    trees.add(temptree)
    tree_timer = pygame.time.get_ticks()
```

Car Class

```
class Car(Sprite):
    def __init__(self):
        Sprite.__init__(self)
        self.image = image.load("Car.png")
        self.rect = self.image.get_rect()
        self.x = 540
        self.y = 500
        self.rect.left = self.x
        self.rect.top = self.y

    def move(self,x,y):
        self.x +=x
        self.y +=y
        self.rect.left = self.x
        self.rect.top = self.y
```

This class is used to change the player into a car in driving phase.

It has the function “move” that is used to move the car image.

Example of Car object:

```
#creating sprites
car = cl.Car()
```

Example of move function:

```
#Checks if movement is true for left or right
if moveleft == True:
    if car.x >=8:
        car.move(-1,0)

elif moveright == True:
    if car.x <= 1058:
        car.move(1,0)
```

Class Road

```
class Road(Sprite):
    def __init__(self, x):
        Sprite.__init__(self)
        self.image = image.load("Road.png")
        self.rect = self.image.get_rect()
        self.x = x
        self.y = 0
        self.rect.center = (self.x,self.y)
    def movedown(self):
        self.y += 10
        self.rect.center = (self.x,self.y)
```

This class is used to create roads for the driving phase. The function “movedown” is used to move the road down.

Global Variables in Game_Classes

```
#Colour used
white = (255,255,255)
black = (0,0,0)
light_green = (150,255,150)
dark_green = (5,20,5)
grey = (100,100,100)
red = (255,100,100)
```

These are the colours used in the program.

Explanation of Functions in Game_Phase.py

Function Start_Menu()

```
#Startmenu function to be called
def Start_Menu():

    #Set the variables and displays
    main_frame = display.set_mode((1080,720))
    breaker = True

    #creating buttons
    start_button = cl.Button("Press to Start", 540,360)
    help_button = cl.Button("How to Play", 540,440, backcolor = cl.light_green)

    #grouping sprites
    start_menu_sprites = Group(start_button, help_button)

    #main startmenu loop
    while breaker:
        main_frame.fill(cl.white)
        start_menu_sprites.draw(main_frame)
        display.update()

        #checks if quit button is clicked
        for ev in event.get():
            if ev.type == QUIT:
                pygame.quit()
                exit()

            #checks if start button is clicked
            elif ev.type == MOUSEBUTTONDOWN:
                if start_button.rect.collidepoint(mouse.get_pos()):
                    breaker = False

                if help_button.rect.collidepoint(mouse.get_pos()):
                    help_menu()
```

This function is used to show the starting menu for the game. It has two buttons. One is to go into the help menu and the other is to start the game. If the start button is pressed, it will break the menu and move on to the Main_Game_Display().

Function Main_Game_Display()

```
#defining main game
def Main_Game_Display():

    #Set the displays
    pygame.display.set_caption("Tree Cutting Simulation")
    main_frame = display.set_mode((1080,720))
    breaker = True
    main_game_background = cl.BackGround("MainGameBackground_V2.png")
    main_game_backgroundup = cl.BackGround("MainGameBackgroundup_V1.png")
    #height = 98, width = 1080

    #set the variables
    global log, money, initial_time, total_log, total_money, rates, housestage, cut_timeneeded, car
    log = 10
    money = 0
    total_log = 0
    total_money = 0
    cut_progress = 0
    rates = 100
    housestage = 0
    cut_timeneeded = 60
    car = False

    #creating buttons and sprites
    market_button = cl.Button("Market", 50, 45)
    stats_button = cl.Button("Stats", 375,45)
    upgrade_button = cl.Button("Upgrade", 750,45)
    drive_button = cl.Button("Nocturnal Drive", 250,80)
    normal_button = cl.Button("Regular Drive", 750,80)
    go_home_button = cl.Button("Go Home", 1000,45)
    player = cl.Player()
```

This function is used as the main game. It acts as both a menu and game. Variables, backgrounds, buttons and other objects are set first.

```
#grouping sprites
main_game_sprites = Group(main_game_background, main_game_backgroundup, player,
    market_button, stats_button, upgrade_button, go_home_button)
trees = Group()

#taking initial times
initial_time = pygame.time.get_ticks()
tree_timer = pygame.time.get_ticks()
```

This part is used to create a group for the trees, buttons, backgrounds and the player. Initial_time and tree_timer is set as well.

```
#setting movement as false
moveleft = False
moveup = False
movedown = False
moveright = False
swingmode = False

#setting mini menu as false
showmarketoptions = False

#creating clock
ingameclock = pygame.time.Clock()
```

This part is used to set all variables used to check input to false. It also sets the ingameclock as a Clock().

```
#Main game loop
while breaker:

    #setting frame per second cap to 60
    ingameclock.tick(60)

    #main event checking
    for ev in event.get():
        if ev.type == QUIT:
            pygame.quit()
            exit()
```

This is the main loop of the function. Ingameclock.tick(60) is used to limit the frame per second to 60. The for loop is used as an event checker. The first event checked is if the event is equal to QUIT, the program will close.

```
#checks if market is pressed
elif ev.type == MOUSEBUTTONDOWN:

    #checks if car is present when market is pressed
    if market_button.rect.collidepoint(mouse.get_pos()):
        if car == True and showmarketoptions == False:
            main_game_sprites.add(drive_button)
            main_game_sprites.add(normal_button)
            showmarketoptions = True

        elif car == True and showmarketoptions == True:
            main_game_sprites.remove(drive_button)
            main_game_sprites.remove(normal_button)
            showmarketoptions = False

        else:
            MarketMenu()

    #Checks if stats is pressed
    if stats_button.rect.collidepoint(mouse.get_pos()):
        StatsMenu()

    #checks if upgrade is pressed
    if upgrade_button.rect.collidepoint(mouse.get_pos()):
        UpgradeMenu()

    if drive_button.rect.collidepoint(mouse.get_pos()):
        result = Driving_Phase()
        if result == True:
            MarketMenu(result)

    if normal_button.rect.collidepoint(mouse.get_pos()):
        MarketMenu()

    if go_home_button.rect.collidepoint(mouse.get_pos()):
        House()
```

This part is used to check for mousebutton events. If a mousebutton event is inputted, it will check if the mouse is on any of the buttons and will run what the button is intended to do. For example, if the upgrade button is clicked on, it will run UpgradeMenu() which is responsible for the upgrade display. For the market, the program will check first if the car variable is True. If it is true, then it will show more options such as nocturnal drive or regular drive. If it is false, it will just run MarketMenu().

```
#checks if button is being pressed
elif ev.type == KEYDOWN:
    if ev.key == K_LEFT:
        moveleft = True
        player.direction("left")

    elif ev.key == K_RIGHT:
        moveright = True
        player.direction("right")

    elif ev.key == K_DOWN:
        movedown = True

    elif ev.key == K_UP:
        moveup = True

    elif ev.key == K_z:
        swingmode = True

#checks if button is let go
elif ev.type == KEYUP:

    if ev.key == K_LEFT:
        moveleft = False

    elif ev.key == K_RIGHT:
        moveright = False

    elif ev.key == K_DOWN:
        movedown = False

    elif ev.key == K_UP:
        moveup = False

    elif ev.key == K_z:
        swingmode = False
```

This part checks if any of the arrow keys or the z key are pressed. If they are pressed, it will set the respective variables as True. It also makes it so that the character faces left or right respective to the key pressed. If any of the arrow keys or the z key is let go, the respective variables will turn False.

```
#Checks if movement is true and direction
if moveleft == True:
    if player.x >= 8:
        player.move(-5,0)

if moveright == True:
    if player.x <= 1058:
        player.move(5,0)

if movedown == True:
    if player.y <= 574:
        player.move(0,5)

if moveup == True:
    if player.y >= 102:
        player.move(0,-5)

#checks if player is swinging
if swingmode == True:
    player.swingstance()
    for temptree in trees:
        if player.rect.colliderect(temptree):
            if cut_progress >= cut_timeneeded:
                trees.remove(temptree)
                log += 1
                total_log += 1
                cut_progress = 0
            else:
                cut_progress += 1

if swingmode == False:
    player.defaultstance()
```

This part of the code checks if any of the variables above turns true. If move for a direction is true, the player sprite will move with respect to the direction. If swingmode is true, it will set the player into swing mode. It then will check if the player is touching a sprite from the trees group. If the player collides, it will increase the cut_progress until the cut_progress is greater or equal to cut_timeneeded. Once it is true, it will remove the tree, increase log and total log by one and reset cut progress. It will also check if swingmode is false. If it is false, it will set the player back to default stance.

```
#checks if it is 3 seconds and spawns a tree
if pygame.time.get_ticks() - tree_timer >= 3000:
    temptree = cl.Tree()
    trees.add(temptree)
    tree_timer = pygame.time.get_ticks()

for sprites in main_game_sprites:
    if isinstance(sprites,cl.Button):
        if sprites.rect.collidepoint(mouse.get_pos()):
            sprites.change_colour(cl.white)

        else:
            sprites.change_colour(cl.black)


#updating screen
main_frame.fill(cl.light_green)
trees.draw(main_frame)
main_game_sprites.draw(main_frame)
display.update()
```

This part of the code spawns a tree every three seconds by creating a tree object and adding it to the trees group. It also checks if the mouse is hovering over sprites that are button class. If the mouse is hovering over a button sprite, it will change the button font colour into white. If the mouse is not hovering, it will change the colour back to black. At the end, it fills the screen with light green, draws trees and main game sprites and displays update.

Function MarketMenu()

```
defining menu for Market button
ef MarketMenu(result = None):
    global money, log, total_money, rates

    #creating display
    pygame.display.set_caption("Tree Cutting Simulation")
    main_frame = display.set_mode((1080,720))

    #creating background and buttons
    main_game_background = cl.BackGround("MainGameBackground_V2.png")
    market_background = cl.BackGround("Market_Background V1.png")

    #creating buttons
    sell_button = cl.Button("Sell All", 540,550,120)
    back_button = cl.Button("Back to Game", 100,150,42, cl.white)
    money_button = cl.Button("Money: %d"%money, 300,450,50)
    log_button = cl.Button("Log: %d"%log, 700,450,50)
    if result:
        doublerate = rates*2
        rates_button = cl.Button("Rates: %d"%doublerate, 850,150,50, color = cl.red)
    else:
        rates_button = cl.Button("Rates: %d"%rates, 850,150,50, color = cl.white)
```

MarketMenu() is used to show the menu for the game market. The market is used as a place to convert logs into money and is the main way of earning money. The function has a parameter result with default value None. This is used to check if the player successfully won the Driving_Phase(). If the player did, it will temporarily double the rate. If the player did not take part in Driving_Phase(), it will use the standard rate.

```
#Main market loop
breaker = True
while breaker:

    #creating and updating screen
    main_frame.fill(cl.light_green)
    market_sprites.draw(main_frame)
    display.update()

    #main event checking
    for ev in event.get():
        #checks if quit button is clicked
        if ev.type == QUIT:
            pygame.quit()
            exit()

        if ev.type == MOUSEBUTTONDOWN:
            #checks if back button is clicked
            if back_button.rect.collidepoint(mouse.get_pos()):
                breaker = False

            #checks if sell button is clicked
            if sell_button.rect.collidepoint(mouse.get_pos()):
                if result:
                    money += log * (rates * 2)
                    total_money += log * (rates * 2)
                    log = 0
```

```
                else:
                    money += log * rates
                    total_money += log * rates
                    log = 0

#updates money and logs
money_button.update_message("Money: %d"%money)
log_button.update_message("Log: %d"%log)
```

This part of the function is used to loop the display and to check events. If back button is clicked, it will break the loop. If sell_button is clicked, it will check if result is true or not. If the result is true, rates are doubled. If not, it will use standard rates. Then, it will convert all logs into money depending on the rate.

Function StatsMenu()

```
def StatsMenu():

    #Set the displays
    pygame.display.set_caption("Tree Cutting Simulation")
    main_frame = display.set_mode((1080,720))
    breaker = True
    main_game_background = cl.BackGround("MainGameBackground_V2.png")
    main_game_backgroundup = cl.BackGround("MainGameBackgroundup_V1.png")
    #height = 98, width = 1080

    #setting variables
    global log, money, initial_time, total_log, total_money, rates, housestage, cut_timeneeded, car


    time_spent = (pygame.time.get_ticks() - initial_time) / 1000
    minutes_spent = time_spent/60
    seconds_spent = time_spent%60
    hours_shown = minutes_spent/60
    minutes_shown = minutes_spent%60
```

```
#creating buttons
back_button = cl.Button("Back to Game", 100,150,42, backcolor = cl.light_green)
total_money_button = cl.Button("Total Earned: %d"%total_money, 540, 150,42)
total_log_button = cl.Button("Total Trees Cut: %d"%total_log, 540, 200, 42)
total_time_button = cl.Button("Time Spent: %dhours %dminutes %dseconds"%(hours_shown, minutes_shown, seconds_spent), 540, 250, 42)
housestagenum = cl.Button("Current House Stage: %d"%housestage, 540, 300,42)
save_button = cl.Button("Save Game", 100,550,42)
load_button = cl.Button("Load Game", 100,600,42)

#grouping sprites
stats_shown = Group(main_game_background, main_game_backgroundup, total_money_button, total_log_button, total_time_button,
    housestagenum,)

stats_buttons = Group(back_button, save_button, load_button)
```

This function is used to display a screen showing statistics of the game such as total number of logs earned or time spent.

This part of the code is used to set variables such as log or time_spent so that the buttons can show the numbers. It will also be used to save or load game progress.

```
#Main stats menu loop
while breaker:

    #creating and updating screen
    main_frame.fill(cl.light_green)
    stats_shown.draw(main_frame)
    stats_buttons.draw(main_frame)
    display.update()

    #main event checking
    for ev in event.get():
        #checks if quit is pressed
        if ev.type == QUIT:
            pygame.quit()
            exit()

        if ev.type == MOUSEBUTTONDOWN:
            #checks if back button is pressed
            if back_button.rect.collidepoint(mouse.get_pos()):
                breaker = False

            #checks if save button is pressed
            if save_button.rect.collidepoint(mouse.get_pos()):
                save_varlist = [log,money,initial_time,total_log,total_money,rates,housestage,cut_timeneeded,car]

                tempfile = open("Save.txt", "w")

                #writes every variable in a text file
                for data in save_varlist:
                    tempfile.write("%s\n" %str(data))

                tempfile.close()
```

This part is used to loop the display. It is also used to check event. If event is QUIT, it will exit the code. If event is mouse click on back button, it will break the loop. If event is mouse click on save_button, it will open a file called “Save.txt” and write down all the important variables inside it.

```
#checks if load button is pressed
if load_button.rect.collidepoint(mouse.get_pos()):
    tempfile = open("save.txt")
    tempsavelist = tempfile.read().split("\n")

    #set all the variables to the ones in the file
    log = int(tempsavelist[0])
    money = int(tempsavelist[1])
    initial_time = int(tempsavelist[2])
    total_log = int(tempsavelist[3])
    total_money = int(tempsavelist[4])
    rates = int(tempsavelist[5])
    housestage = int(tempsavelist[6])
    cut_timeneeded = int(tempsavelist[7])
    car = tempsavelist[8]

    tempfile.close()
```

If event is mouse click on load_button, it will open the “Save.txt” file and replace variables with the data saved inside the file.

```
#checks if buttons in group stats_button is hovered
for sprite in stats_buttons:
    if sprite.rect.collidepoint(mouse.get_pos()):
        sprite.change_colour(cl.white)
    else:
        sprite.change_colour(cl.black)

#update time spent
minutes_spent = time_spent/60
seconds_spent = time_spent%60
hours_shown = minutes_spent/60
minutes_shown = minutes_spent%60
time_spent = (pygame.time.get_ticks() - initial_time) / 1000
```

```
#updates buttons
total_time_button.update_message("Time Spent: %dhours %dminutes %dseconds"%(hours_shown, minutes_shown, seconds_spent))
total_money_button.update_message("Total Earned: %d"%total_money)
total_log_button.update_message("Total Trees Cut: %d"%total_log)
total_time_button.update_message("Time Spent: %dhours %dminutes %dseconds"%(hours_shown, minutes_shown, seconds_spent))
housestagenum.update_message("Current House Stage: %d"%housestage)
```

This part of the code is used to check if the mouse is hovering over the sprites in stats_button. If it is, it will change font colour to white. If it is not, it will change font colour to black.

It is also used to update the time spent and the buttons.

Function UpgradeMenu()

```
#menu for upgrades
def UpgradeMenu():
    global money, log, rates, housestage, cut_timeneeded, rates, car

    #creating display
    pygame.display.set_caption("Tree Cutting Simulation")
    main_frame = display.set_mode((1080,720))

    #creating background and buttons
    main_game_background = cl.BackGround("MainGameBackground_V2.png")
    upgrade_background = cl.BackGround("Upgrade_Background V1.png")

    #setting variables
    purchaseable_stage = housestage + 1
    houseprice = purchaseable_stage * 500
    rate_price = rates*2.5
    purchaseable_rate = rates+10
    carprice = 1000
    purchaseable_speed = cut_timeneeded - 10
    if cut_timeneeded <= 0:
        speedprice = 0
    else:
        speedprice = 48000/cut_timeneeded

    #creating buttons
    back_button = cl.Button("Back to Game", 100,50,42, cl.white)
    money_button = cl.Button("Money: %d"%money, 900,125,50, color = cl.white)
    log_button = cl.Button("Log: %d"%log, 900,175,50, color = cl.white)
    buy_house_button = cl.Button("Buy: %d"%houseprice, 120, 550, 42, color = cl.white)
    buy_house = cl.Button("House Stage %d"%purchaseable_stage, 120, 500, 42, color = cl.white)
    buy_rate_button = cl.Button("Buy: %d"%rate_price, 350,550,42,color = cl.white)
    buy_rate = cl.Button("Rates = %d"%purchaseable_rate, 350,500,42,color = cl.white)
    buy_car_button = cl.Button("Buy: %d"%carprice, 600,550,42,color = cl.white)
    buy_car = cl.Button("Car", 600,500,42,color = cl.white)
    buy_speedup = cl.Button("Cut Speed: %d Ticks"%purchaseable_speed, 850,500,42,color = cl.white)
    buy_speedup_button = cl.Button("Buy: %d"%speedprice, 850,550,42,color = cl.white)
```

```
#grouping sprites
upgrade_sprites = Group(main_game_background, upgrade_background, back_button, money_button, log_button, buy_house,
    buy_house_button, buy_rate_button, buy_rate, buy_car_button, buy_car, buy_speedup, buy_speedup_button)
```

This function is used to show the upgrade menu to allow the user to buy upgrades.

This part of the function is used to set variables, buttons and sprite groups.

```
#Main market loop
breaker = True
while breaker:

    #creating and updating screen
    main_frame.fill(cl.light_green)
    upgrade_sprites.draw(main_frame)
    display.update()

    #main event checking
    for ev in event.get():
        #checks if quit button is clicked
        if ev.type == QUIT:
            pygame.quit()
            exit()
```

This part of the function is used to loop the display and to check events.

If the event is QUIT, it will exit.

```
#checks for event: mouse button
if ev.type == MOUSEBUTTONDOWN:
    if back_button.rect.collidepoint(mouse.get_pos()):
        breaker = False

    if buy_house_button.rect.collidepoint(mouse.get_pos()):
        if money >= houseprice and housestage < 5:
            money -= houseprice
            housestage += 1
            houseprice = (housestage + 1) * 500
            purchaseable_stage += 1

    if buy_rate_button.rect.collidepoint(mouse.get_pos()):
        if money >= rate_price:
            money -= rate_price
            rates += 10
            purchaseable_rate += 10
            rate_price = rates * 2.5

    if buy_car_button.rect.collidepoint(mouse.get_pos()):
        if money >= carprice and (car == False or car == "False"):
            money -= carprice
            car = True

    if buy_speedup_button.rect.collidepoint(mouse.get_pos()):
        if money >= speedprice and cut_timeneeded > 0:
            money -= speedprice
            cut_timeneeded -= 10
            purchaseable_speed = cut_timeneeded - 10
            if cut_timeneeded <=0:
                speedprice = None
            else:
                speedprice = 48000/cut_timeneeded
```

This part of the code is used to check events when the user clicks the mouse buttons. If back_button is clicked, it will break the loop. If any of the buy buttons are clicked, it will check if the player has enough money to buy the upgrade. If there is enough money, it will decrease money with the price of the upgrade and undergo the upgrade such as increase the rate.

```
#updates button
money_button.update_message("Money: %d"%money)
log_button.update_message("Log: %d"%log)
buy_rate_button.update_message("Buy: %d"%rate_price)
buy_rate.update_message("Rates = %d"%purchaseable_rate)


#updates housestage button and caps the housestage to 5
if housestage < 5:
    buy_house.update_message("House Stage %d"%purchaseable_stage)
    buy_house_button.update_message("Buy: %d"%houseprice)

else:
    buy_house.update_message("House Completed")
    buy_house_button.update_message("Bought")

#updates buy for car into bought if purchased
if car == True:
    buy_car_button.update_message("Bought")

if cut_timeneeded > 0:
    buy_speedup.update_message("Cut Speed: %d Ticks"%purchaseable_speed)

    buy_speedup_button.update_message("Buy: %d"%speedprice)

else:
    buy_speedup.update_message("Cut Speed: Maxed")
    buy_speedup_button.update_message("Bought")
```

This part updates buttons and checks if any of the upgrades are already maxed. If the car variable is True, it will show bought instead of price and the player may not buy the car again. This also applies to speed upgrade and house upgrade.

Function Driving_Phase()

```
#driving phase
def Driving_Phase():

    #Set the displays
    pygame.display.set_caption("Tree Cutting Simulation")
    main_frame = display.set_mode((1080,720))
    breaker = True
    main_game_background = cl.BackGround("MainGameBackground_V2.png")
    main_game_backgroundup = cl.BackGround("MainGameBackgroundup_V1.png")
    #height = 98, width = 1080

    #creating sprites
    car = cl.Car()

    road = Group()
    driving_phase_sprites = Group(main_game_background, main_game_backgroundup, car)

    #taking initial times
    initial_drivingtime = pygame.time.get_ticks()
    road_timer = pygame.time.get_ticks()
```

```
    #setting movement as false
    moveleft = False
    moveright = False

    #creating clock
    ingameclock = pygame.time.Clock()

    #creating first road coordinate
    tempxcoor = 540

    #declaring that car is on road
    onroad = True

    #loads and plays music
    pygame.mixer.music.load("Deja Vu Initial D.mp3")
    pygame.mixer.music.play()
```

This function is used to run the driving part of the game. This part of the code is used to create variables such as onroad, objects such as car and sprite groups such as road. The mixer.music is also loaded and played here.

```
#Main stats menu loop
while breaker:

    #capping fps to 60
    ingameclock.tick(60)

    #creating and updating screen
    main_frame.fill(cl.dark_green)
    road.draw(main_frame)
    driving_phase_sprites.draw(main_frame)
    display.update()


    #main event checking
    for ev in event.get():
        #checks if quit is pressed
        if ev.type == QUIT:
            pygame.quit()
            exit()
```

This part is used to loop the display, set the frame cap to 60 frame per second and to check for event. If the event is QUIT, it will exit.

```
#checks if button is being pressed
elif ev.type == KEYDOWN:

    if ev.key == K_LEFT:
        moveleft = True

    elif ev.key == K_RIGHT:
        moveright = True

#checks if button is let go
elif ev.type == KEYUP:

    if ev.key == K_LEFT:
        moveleft = False

    elif ev.key == K_RIGHT:
        moveright = False
```

If event is KEYDOWN and left arrow or right arrow, moveleft or moveright will turn True respectively. If event is keyup for left arrow or right arrow, moveleft or moveright will turn False respectively.

```
#Checks if movement is true for left or right
if moveleft == True:
    if car.x >=8:
        car.move(-1,0)

elif moveright == True:
    if car.x <= 1058:
        car.move(1,0)

#creating roads every 0.1 seconds
if pygame.time.get_ticks() - road_timer >= 100:
    sway = random.randint(-10,10)
    tempxcoor = tempxcoor + sway
    temproad = cl.Road(tempxcoor)
    road.add(temproad)
    road_timer = pygame.time.get_ticks()

#makes object move downwards and remove roads that leave the screen
for object in road:
    object.movedown()
    if object.y >= 720:
        road.remove(object)
```

This part checks if moveleft or moveright is True and will move the car left or right accordingly.

It will also check if pygame.time.get_ticks() – road_timer is greater or equal to 100. If it is true, it will randomize the x coordinate of the next road, add the road to the group road and resets the road_timer.

This part also makes the objects in road to run the function movedown which is to move the road down. It also checks if the road's y position is greater or equal to 720. If it is, it will remove the road from the group.

```
#after 2 seconds, game starts to check if car is offroad
if pygame.time.get_ticks() - initial_drivingtime >= 2000:

    if spritecollideany(car, road):

        onroad = True
    else:
        onroad = False

#set winning conditions
if pygame.time.get_ticks() - initial_drivingtime >= 35000:
    breaker = False
    return Success_Menu()
#checks if car is offroad
if onroad == False:
    breaker = False
    return Crash_Menu()
```

This part is used to check if the car goes offroad. After roughly 2 seconds, it will start checking if the car is colliding with any of the objects in the road group. If it is colliding, it will set onroad as True. If it is not on the road, it will set it as False.

If onroad is False, it will break the game and run Crash_Menu().

If pygame.time.get_ticks – initial_drivingtime is greater or equal to 35000, it will break the loop and run Success_Menu()

Function Crash_Menu()

```
#menu when car crashes
def Crash_Menu():
    global log

    #Set the variables and displays
    main_frame = display.set_mode((1080,720))
    breaker = True
    logs_lost = int(log*0.2)
    log -= logs_lost

    #creating buttons
    youcrash_sign = cl.Button("You went offroad!", 540,100,100)
    logslost_sign = cl.Button("You lost %d logs!"%logs_lost, 540, 250)
    continue_button = cl.Button("Continue", 540,360)

    #grouping sprites
    start_menu_sprites = Group(continue_button, youcrash_sign, logslost_sign)

    #main startmenu loop
    while breaker:
        main_frame.fill(cl.white)
        start_menu_sprites.draw(main_frame)
        display.update()

        #checks if quit button is clicked
        for ev in event.get():
            if ev.type == QUIT:
                pygame.quit()
                exit()

            #checks if start button is clicked
            elif ev.type == MOUSEBUTTONDOWN:
                if continue_button.rect.collidepoint(mouse.get_pos()):
                    pygame.mixer.music.stop()
                    breaker = False
                    result = False
                    return result
```

This part is used to show the menu when you crash in the driving_phase. It removes 20% of log. The display loop checks for events such as clicking on the continue button. If the continue button is clicked on, the loop will break and the result False is returned.

Function Success_Menu()

```
def Success_Menu():
    #Set the variables and displays
    main_frame = display.set_mode((1080,720))
    breaker = True

    #creating buttons
    youreach_sign = cl.Button("You reached your destination!", 540,100,100)
    continue_button = cl.Button("Continue", 540,360)

    #grouping sprites
    start_menu_sprites = Group(continue_button, youreach_sign)

    #main startmenu loop
    while breaker:
        main_frame.fill(cl.white)
        start_menu_sprites.draw(main_frame)
        display.update()

        #checks if quit button is clicked
        for ev in event.get():
            if ev.type == QUIT:
                pygame.quit()
                exit()

            #checks if start button is clicked
            elif ev.type == MOUSEBUTTONDOWN:
                if continue_button.rect.collidepoint(mouse.get_pos()):
                    pygame.mixer.music.stop()
                    breaker = False
                    result = True
                    return result
```

This function is used to show the menu when Driving_Phase is completed without crashing. It is almost the same as Crash_Menu with difference of no logs lost and returned result value is True.

Function House()

```
def House():
    #Set the displays
    pygame.display.set_caption("Tree Cutting Simulation")
    main_frame = display.set_mode((1080,720))
    breaker = True
    main_game_background = cl.BackGround("MainGameBackground_V2.png")
    main_game_backgroundup = cl.BackGround("MainGameBackgroundup_V1.png")
    #height = 98, width = 1080

    #check which housestage is active
    if housestage == 0:
        housepicture = cl.BackGround("Home 0.png")

    elif housestage == 1:
        housepicture = cl.BackGround("Home 1.png")

    elif housestage == 2:
        housepicture = cl.BackGround("Home 2.png")

    elif housestage == 3:
        housepicture = cl.BackGround("Home 3.png")

    elif housestage == 4:
        housepicture = cl.BackGround("Home 4.png")

    elif housestage == 5:
        housepicture = cl.BackGround("Home 5.png")
```

This function is used to display a picture of the player's house. This part sets the background image depending on the variable housestage.

```
#Main house menu loop
while breaker:

    #creating and updating screen
    main_frame.fill(cl.white)
    house_menu_sprites.draw(main_frame)
    display.update()

    #main event checking
    for ev in event.get():
        #checks if quit is pressed
        if ev.type == QUIT:
            pygame.quit()
            exit()

        if ev.type == MOUSEBUTTONDOWN:
            #checks if back button is pressed
            if back_button.rect.collidepoint(mouse.get_pos()):
                    breaker = False
```

This part is used to loop the display and check events.

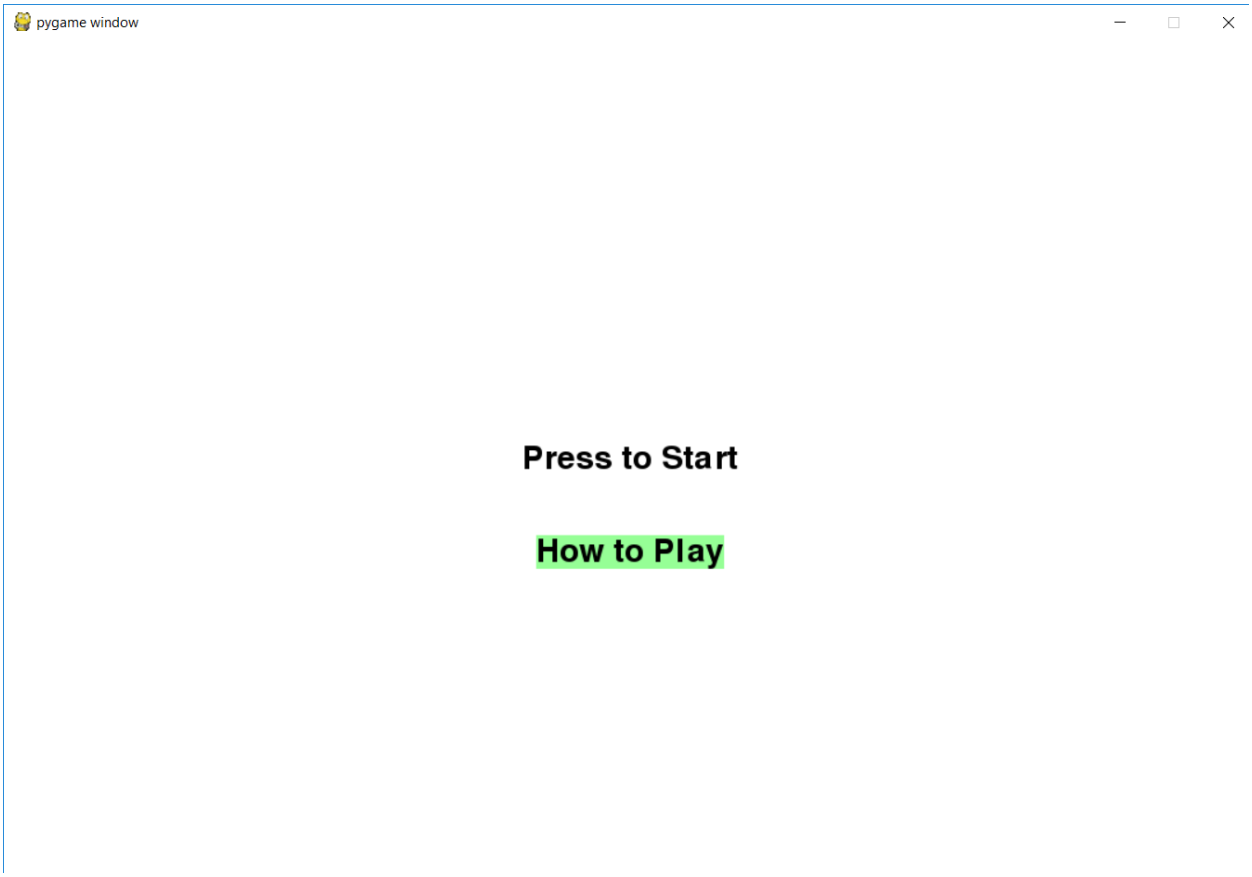
Explanation of Tree_Cutting_Simulation.py

```
from pygame import *
from pygame.sprite import *
from pygame.mixer import *
import Game_Classes as cl
import Game_Phase as gp

pygame.init()
pygame.mixer.init()

#shows start menu
gp.Start_Menu()

#shows game display
gp.Main_Game_Display()
```

This part is used to initialize pygame and to show the Start_Menu() and Main_Game_Display()

In game pictures:

Start Menu:



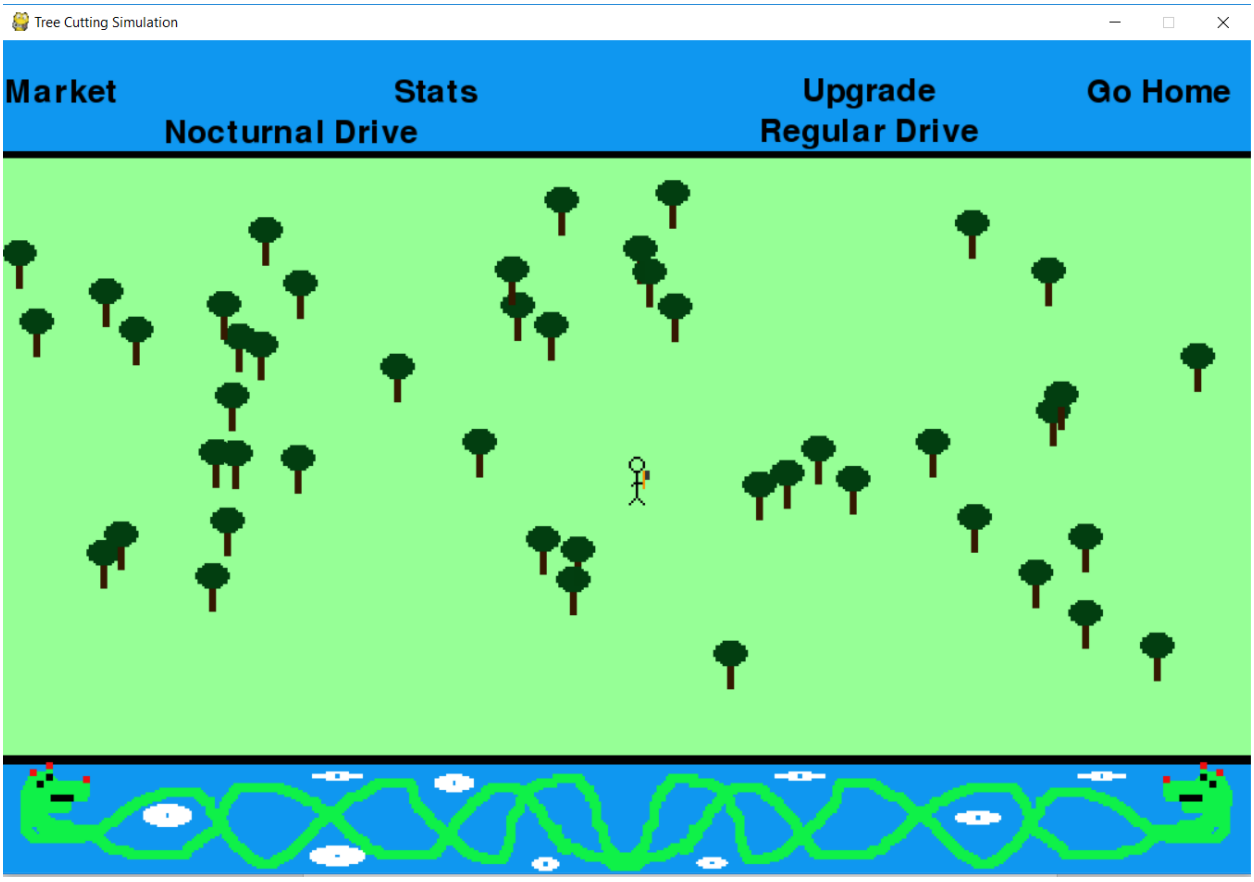
Help Menu:

Controls: Arrow key Up to move up.
Arrow key Down to move down.
Arrow key Left to move left.
Arrow key Right to move right.
Z to initiate cut tree mode.
In Car: Arrow key Up to move up.
Arrow key Down to move down.

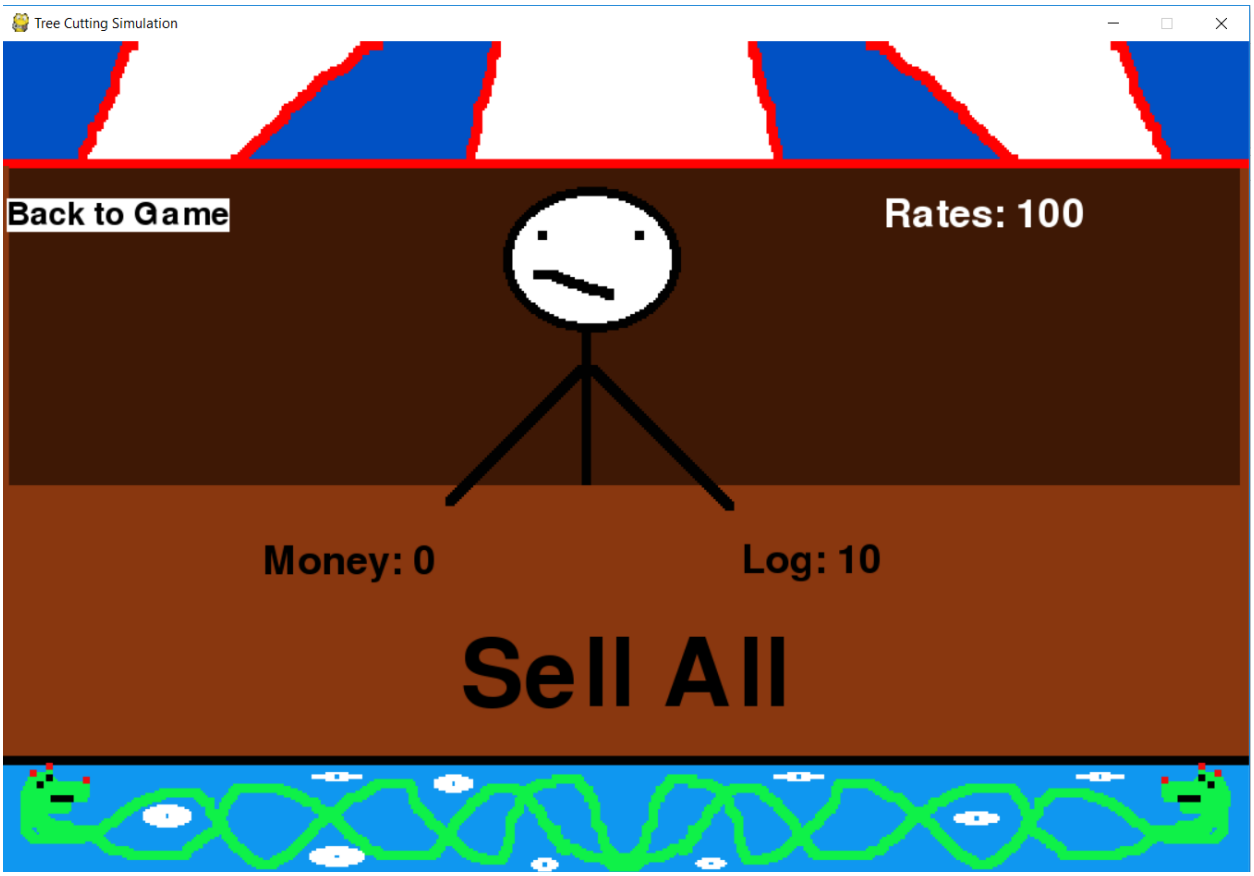
Mousebutton to move through the menu.

Aim: Make as much money as you can and upgrade your equipment and housing.

Main Game Display:

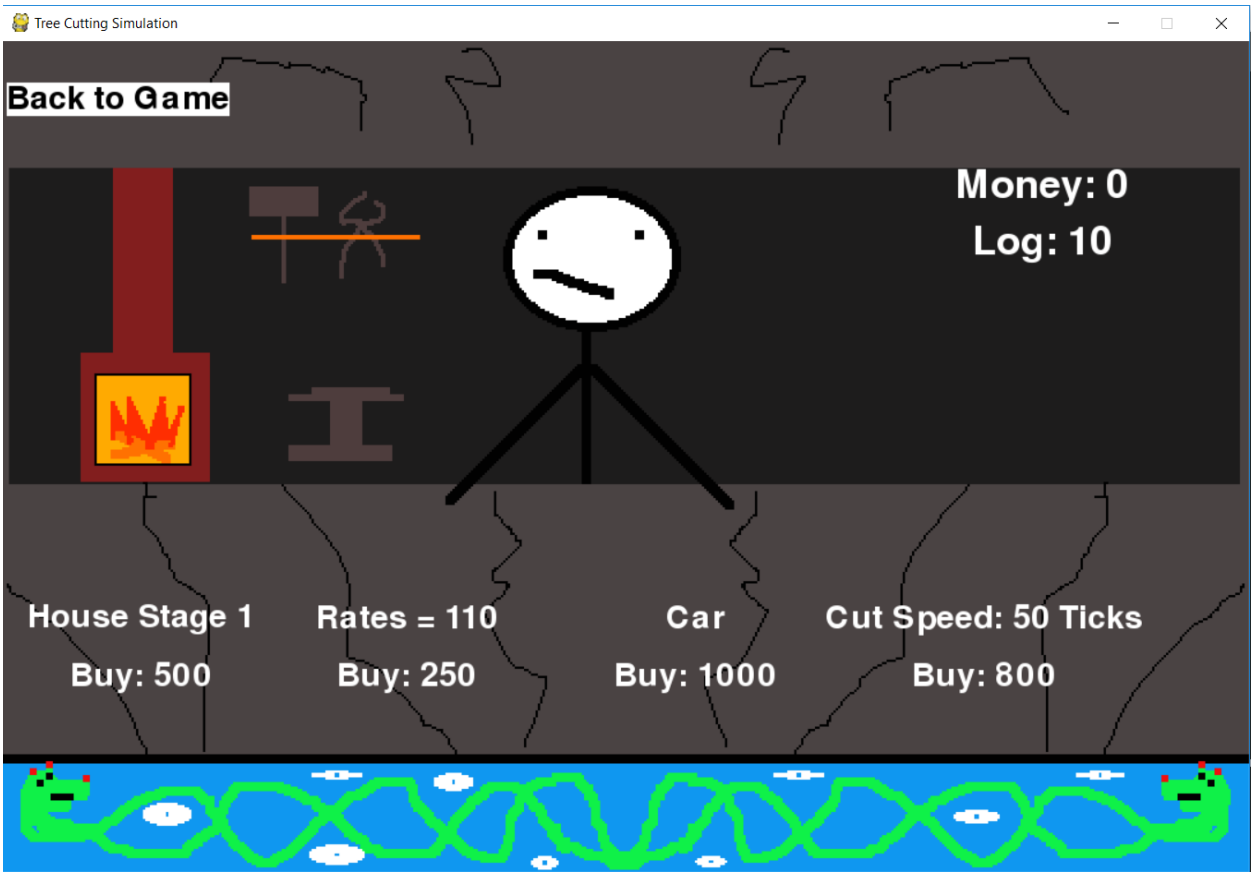


Market Display:

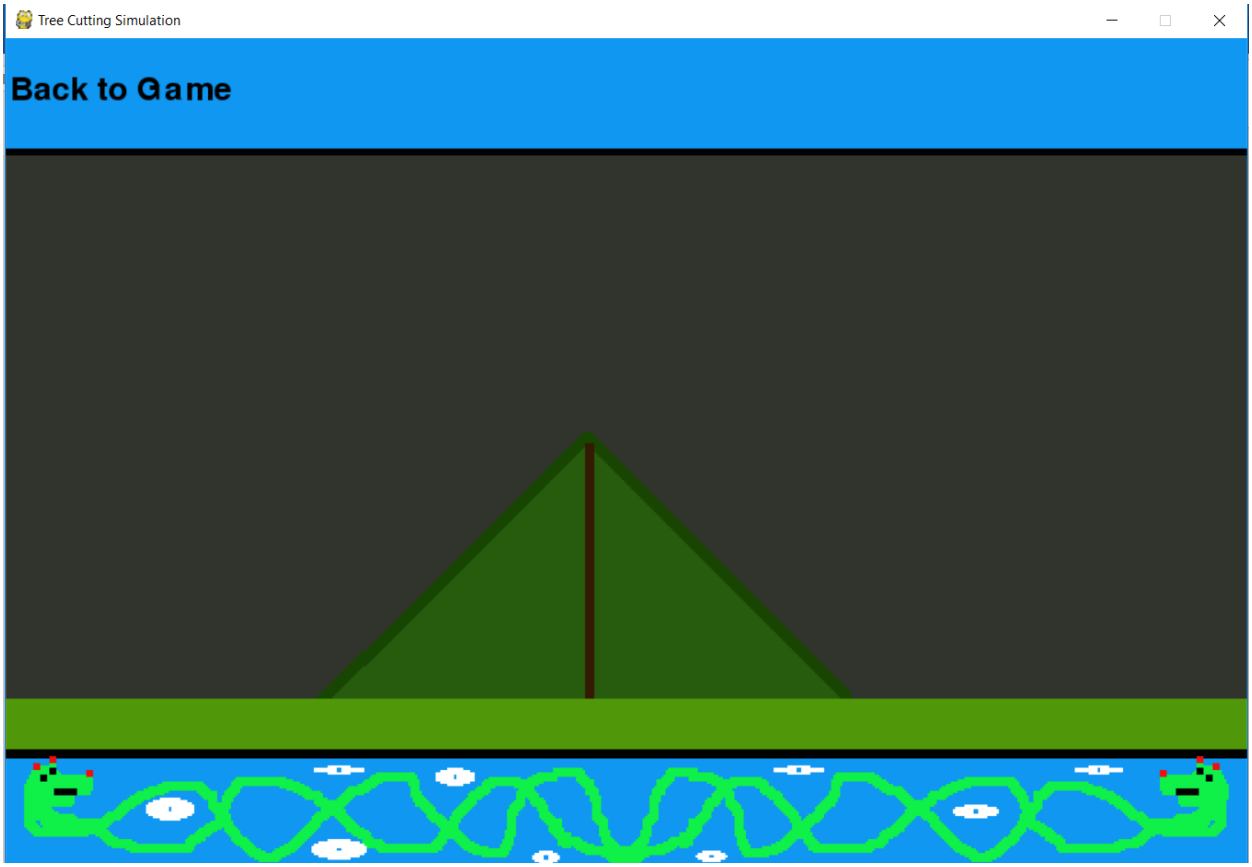


Stats Menu Display:

Upgrade Menu Display:

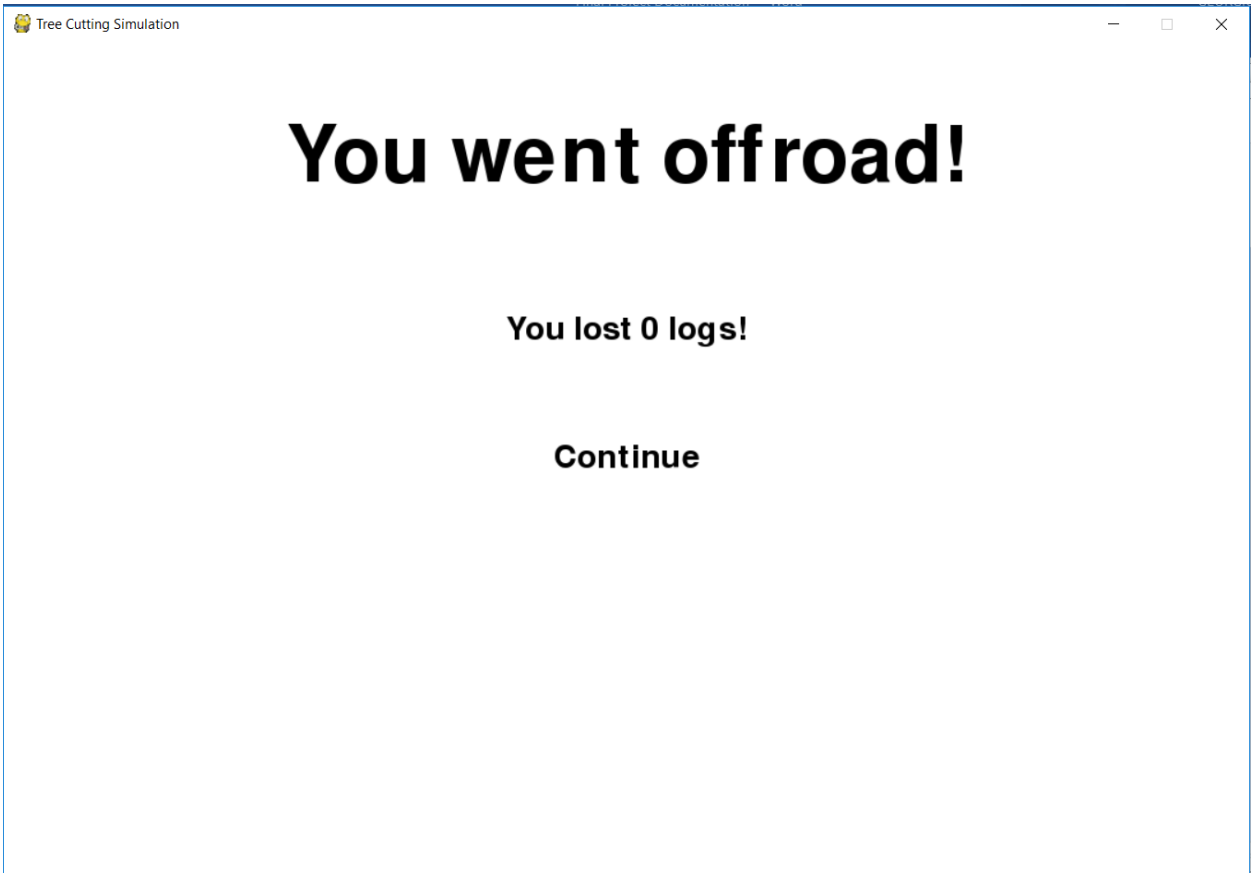


Home Menu Display:

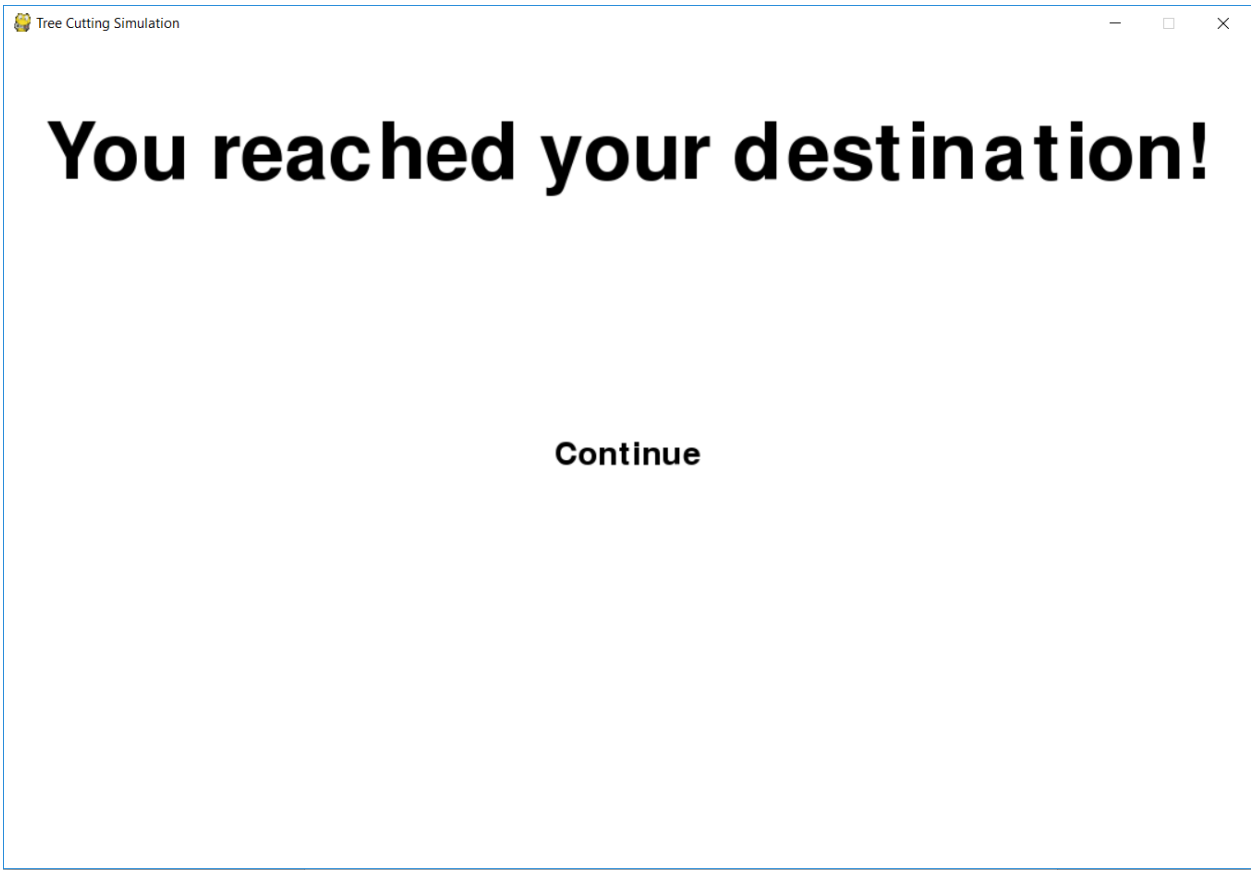


Driving Phase Display:

Crash Menu Display:



Success Menu Display:



Disclaimer:

The music “Déjà vu” is created by Dave Rodgers.