MP2 Documentation

A 2D physics-based arcade game

File structure

engine

|-> hs_handling.py |-> physical_object.py |-> terrain.py |-> text_input.py

|-> resources.py |-> button.py |-> obstacles.py |-> sound_loop.py

|-> custom_sprites.py |-> player.py |-> scrolling_text.py

resources

|-> png files |-> wav files

game.py

*hillclimb.csv

*motor_race.csv

NOTE: *created files by the game for high score saves

Game modes

Hillclimb

The first game mode is a hillclimb type of game that can be won by bringing at least one box to the finish line as fast as possible. The score is computed by the formula $score = 130 - time_{elapsed} + amount_{boxes} * 10$. If there are no boxes left or the score evaluates to a negative number, then it is game over. High scores are ranked in descending order. (Highest to lowest points)

Motorace

The second game mode is always winnable. However, the score is dependent on time thus it is a time trial mode of game. The score is just the evaluated by the time that elapsed. High scores are ranked in ascending order. (Shortes time to longest time)

Obstacles and Terrain generation

Obstacles and terrain are generated using the following equations and its graph in pixel units.

bounds

terrain graph

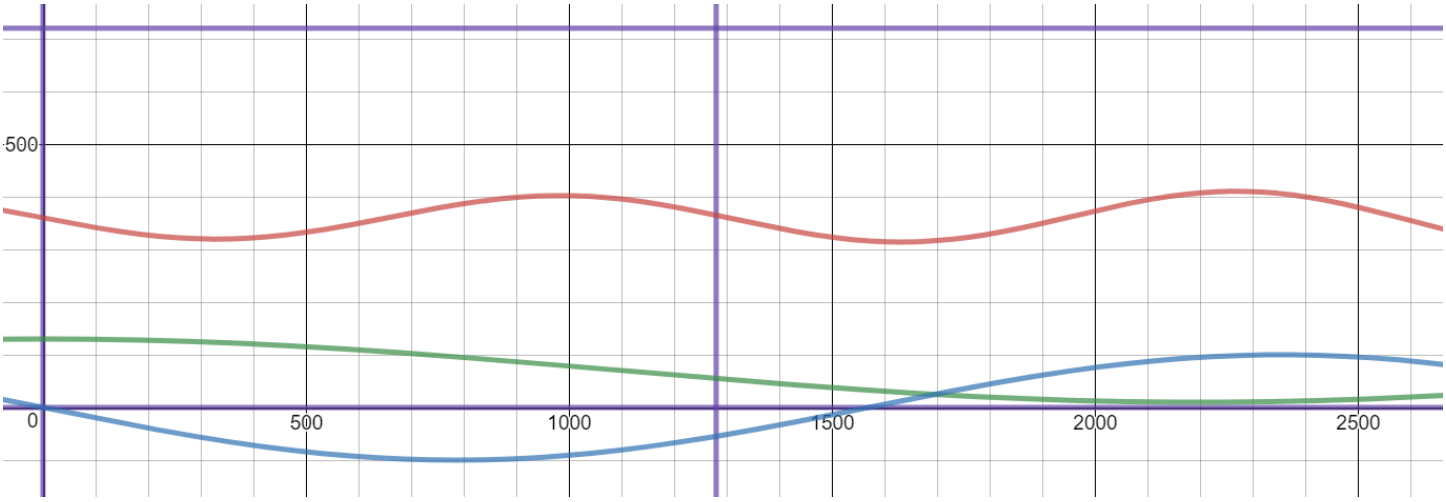
$$360 - 60\sin\left(\frac{x}{200}\right) + 20\sin\left(\frac{x}{185}\right)$$

terrain variation graph

$$70 - 60\sin\left(\frac{x}{700} - \frac{\pi}{2}\right)$$

obstacles graph

$$100\left(-\sin\left(\frac{x+1}{500}\right)\right)$$



The terrain equation is the height in pixels of the terrain given the x-distance in pixels. It is variated using the terrain variation equation multiplied by a Gaussian distribution with mean 0 and sigma that defaults to 0.65.

The obstacles equation is $y = -\sin\left(\frac{x+1}{500}\right)$. For the sake of a more visible graph, the equation was multiplied by 100 to exaggerate. This equation is used to turn on and off the generation of obstacles for a given x value. A positive y value means an obstacle may be generated while a negative y value means it won't generate an obstacle.

External modules

Pyglet (GUI, main loop, event handling)

Pyglet was used for the main structure of the game. It handles the main loop and keyboard and mouse event handling. It also acted as the camera for the physics engine module, pymunk. More information about pyglet is available at https://pyglet.readthedocs.io/. Its source code with examples is available at https://bitbucket.org/pyglet/pyglet/wiki/Home. Also, the module may be installed using the

```
pip install pyglet
```

command. This game works as of pyglet 1.3.2.

Pymunk (Physics engine, physics event handling)

Pymunk is a 2D physics engine that is built on top of c-library, chipmunk. It was made to be “Pythonic” such that it works just like a python library. For this game, it was used for the physics engine only. Sprites and primitive drawing were handled by pyglet. Also, pymunk supports pyglet debugging using the submodule,

```
pymunk.pyglet_util
```

This allows for fast prototyping by handling the drawing in pyglet. However, this is not optimized and should not be used for the final program. More information about pymunk is available at https://www.pymunk.org/. The module may be installed using the

```
pip install pymunk
```

command. This game works as of pymunk 5.4.0.

Engine modules

hs_handling.py

Contains two functions that are used to retrieve and add high scores.

```
get_highscores(filename)
```

This function gets the high score from a file specified by the filename parameter. It returns a None type if the file is not found. Normally, it returns a tuple of tuple(str(name), float(score)).

```
add_highscore(filename, name, score, ascending=True)
```

This function adds a high score to a file specified by the filename parameter. It only stores the top 50. It stores the high score in order specified by the ascending keyworded parameter that defaults to True. The scores are stored in comma separated columns per row. It doesn’t return anything.

resources.py

Loads all the resources (images and sound). Centers the anchor point for all images. Loads all media without streaming.

custom_sprites.py

Contains the sprites with a modified update method.

```
class GoalSprite(object)
    def __init__(self, batch, position, group=None)
```

A custom sprite used for showing the progress of the player in the game.

Pyglet batch object -> batch | tuple(int(), int()) -> position | pyglet ordered group object -> group

```
    def update(self, completion_percent)
```

Updates the sprite according to the percent of completion of the player. Recommended to be called in every update of the game.

Float values with range [0, 1] -> completion_percent

```
class MeterSprite(object)
    def __init__(self, batch, position, group=None)
```

A custom sprite to show a circular meter. Used for speedometer and tachometer in the game.

Pyglet batch object -> batch | tuple(int(), int()) -> position | pyglet ordered group object -> group

```
    def update(self, percent)
```
Updates the sprite according to the percent of completion of the player. Recommended to be called in every update of the game.

Float values with range [0, 1] -> percent

```
class ParallaxBG(object):
    def __init__(self, batch, size, bg_index=0):
```
Creates a parallax background. Window dimensions must be passed into size.

tuple(int(), int()) -> size | int() -> bg_index, used in determining which background set will be used

```
    def update(self, offset=0):
```
Updates the background using the offset parameter. Recommended to be called in every update of the game.

float() -> offset

physical_object.py

```
class PhysicalObject(object):
    def __init__(self, body, sprite, rotation_offset=0)
```
Class that binds the body object in pymunk to a sprite object in pyglet. Converts pymunk coordinates to pyglet coordinates to update the sprite coordinates using the computed coordinates made by pymunk physics.

Pymunk body object -> body | Pyglet sprite object -> sprite | float -> rotation_offset

```
    def update(self, x_offset=0)
```
Updates the pyglet sprite coordinates using the computed pymunk body coordinates. Recommended to be called in every update of the game.

Float -> x_offset

button.py

```
class Button(pymunk.Body):
    def __init__(self, batch, id, position, img, dimensions,
                 body_type=pymunk.Body.DYNAMIC, sensor=False,
                 group=pyglet.graphics.OrderedGroup(2)):
```
Used to create physical buttons or GUI buttons.

Pyglet batch object -> batch | float, int, or string -> id | float(int(), int()) -> position | pyglet image resource object -> img

[int() -> dimensions, to create a circular button or tuple(int(), int()) -> dimensions, to create a rectangular button]

pymunk body type constant -> body_type

boolean -> sensor, a True value makes the button to ignore collisions (not physical) else False will make it a physical object

pyglet ordered group object -> group

```
    def update(self, x_offset=0):
```
Updates the pyglet sprite coordinates using the computed pymunk body coordinates. Recommended to be called in every update of the game.

Float -> x_offset

player.py

```
class Vehicle(object):
    def __init__(self, batch, space, window, position, side='left',
                 torque=1000, speed=2*pi):
```

Parent class for vehicles. Not recommended to be instantiated by itself.

Pyglet batch object -> batch | pymunk space object -> space | pyglet window -> window | tuple(int(), int()) -> position | ['left' or 'right'] -> side, left makes the vehicle face towards the right and vice versa | float() -> torque | float() -> speed

```
    def update(self, x_offset=0):
```

Updates all the physical objects in the vehicle. Also updates the engine sound pitch using the motor speed. Recommended to be called in every update of the game.

```
    def forward(self):
```

Makes the vehicle move forward.

```
    def reverse(self):
```

Makes the vehicle move backward.

```
    def stop(self):
```

Turns off torque.

```
class Tank(Vehicle):
    name = 'tank'
    def __init__(self, batch, space, window, position, side='left',
                 add_boxlives=False, torque=300000, speed=40*pi, group=None):
```

Creates a tank type vehicle.

Pyglet batch object -> batch | pymunk space object -> space | pyglet window -> window | tuple(int(), int()) -> position | ['left' or 'right'] -> side, left makes the vehicle face towards the right and vice versa | boolean -> add_boxlives, adds lives if True| float() -> torque | float() -> speed | pyglet ordered group object -> group

```
class MotorBike(Vehicle):
    name = 'motorbike'
    def __init__(self, batch, space, window, position,
                 torque=120000, speed=11*pi, group=None):
```

Creates a motorbike vehicle.

Pyglet batch object -> batch | pymunk space object -> space | pyglet window -> window | tuple(int(), int()) -> position | float() -> torque | float() -> speed | pyglet ordered group object -> group

terrain.py

```
class Terrain(object):
    def __init__(self, batch, space, window, interval=100, mid_height=360,
                 height_change=0.65, end_coordinate=30000, color_set='green',
                 group=None):
```

Generates a terrain using equations with modifiable parameters.

Pyglet batch object -> batch | pymunk space object -> space | pyglet window -> window | Int() -> interval, the length of each terrain shape | int() -> mid_height, the mean height of the terrain | float() -> height_change, the higher the value, the more variated. May be unstable in very high values | int() -> end_coordinate, the length of the whole terrain | str()['green' or 'gray'] -> color_set | pyglet ordered group -> group

```
    def update(self, x_offset=0, update_all=False):
```
Updates the coordinates of the pyglet primitives using the coordinates in the computed pymunk coordinates.

obstacles.py

```
class Obstacles(object):
    def __init__(self, batch, space, window, end_coordinate=30000, radius_range=(10, 40),
                 mass=1, frequency=125, amount=4, x_offset=0, mid_height=380,
                 group=None):
```
Generates obstacles (rocks) using equations with modifiable parameters.

Pyglet batch object -> batch | pymunk space object -> space | pyglet window -> window |
int() -> end_coordinate, the length of the whole terrain | tuple(int(), int()) -> radius_range, the range of radius for the rocks | float() -> mass | int() -> frequency, higher the value, the more frequent a spawn location made |
int() -> amount, the maximum number of rocks per spawn location | int() -> x_offset, to offset the creation of obstacles|
int() -> min_height, the lowest y coordinate for an obstacle to be spawned | pyglet ordered group -> group

```
    def update(self, x_offset=0):
```
Updates the coordinates of the pyglet primitives using the coordinates in the computed pymunk coordinates.

scrolling_text.py

```
class ScrollingText(pyglet.text.layout.ScrollableTextLayout):
    def __init__(self, batch, position, width, height, text='',
                 align='left', timeout=60*3, font_size=20, scroll_speed=1,
                 group=None):
```
Creates an automatically scrolling multiline text to display long information. Used to display the high scores for this game.

Pyglet batch object -> batch | tuple(int(), int()) -> position | int() -> width | int() -> height | str() -> text |
['left' or 'right'] -> align, alignment of the text in the box | int() -> timeout, the number of milliseconds before the scrolling start and restarts | int() -> fonr_size | int() -> scroll_speed | pyglet ordered group -> group

```
    def update_text(self, text):
```
Updates the text to be displayed.

```
    def update(self):
```
Updates the GUI. Recommended to be called in every update of the game.

text_input.py

```
class TextInput(object):
    def __init__(self, batch, position, width, font_size=14,
                 group=None):
```
Creates a box that accepts a long string of user input. Used to get the name to be used in storing the high score.

Pyglet batch object -> batch | tuple(int(), int()) -> position | int() -> width | int() -> font_size | pyglet ordered group -> group

sound_loop.py

```
class SoundLoop(pyglet.media.Player):
    def __init__(self, sound):
```
Creates a looping pyglet media player given a media resource object.

Main module (game.py)

```
class Window(pyglet.window.Window):
    def __init__(self, width, height, caption='', resizeable=False):
```

The main class that runs the game. Contains the main update function and the on_draw event.

```
class GameState(object):
    id = 0
    def __init__(self, batch, space, window, mouse_hover=False, bounded=False):
```

The parent class for all game states.

mouse_hover enables or disables mouse hover events | bounded enables a bounded game area or not (open).

```
class Menu(GameState):
    id = 1
    def __init__(self, batch, space, window):
        super().__init__(batch, space, window, mouse_hover=True, bounded=True)
```

Menu game state. The first game state that is instantiated by the Window class. Creates the main menu of the game using the engine modules. Contains its event handlers and its update method called by the main update function in the Window class.

```
class Game1(GameState):
    id = 4
    name = 'hillclimb'
    def __init__(self, batch, space, window):
        super().__init__(batch, space, window)
```

The first game mode. Instantiated by the Window class triggered by the Menu class. Creates the first game mode of the game using the engine modules. Contains its event handlers and its update method called by the main update function in the Window class.

```
class Game2(GameState):
    id = 5
    name = 'motor_race'
    def __init__(self, batch, space, window):
        super().__init__(batch, space, window)
```

The second game mode. Instantiated by the Window class triggered by the Menu class. Creates the second game mode of the game using the engine modules. Contains its event handlers and its update method called by the main update function in the Window class.

```
class HighScore(GameState):
    id = 3
    def __init__(self, batch, space, window, *args, **kwargs):
        super().__init__(batch, space, window, mouse_hover=True, bounded=True)
```

The high score game state. Instantiated by the Window class triggered by the Menu class. Shows the high scores of a game mode using the engine modules. Contains its event handlers and its update method called by the main update function in the Window class.

```
class Endgame(GameState):
    id = 2
    def __init__(self, batch, space, window, *args, **kwargs):
        super().__init__(batch, space, window, mouse_hover=True, bounded=True)
```

The endgame game state. Instantiated by the Window class triggered by the any of the game mode class [Game1 or Game2]. Shows the conclusion of the game using the engine modules. Contains its event handlers and its update method called by the main update function in the Window class.

Resources

- Pyglet and pymunk documentation found at https://pyglet.readthedocs.io/ and https://www.pymunk.org/ respectively.
- Royalty free looping background music (Breeze) found at https://www.youtube.com/watch?v=5GfW9eIGcpY.
- Sidescroller tileable background set made by https://bevouliin.com/ found at https://opengameart.org/users/bevouliincom. Used under the **CC0 1.0 Universal (CC0 1.0) Public Domain Dedication**. (no copyright)
- Pymunk physics in pyglet tutorial by Attila Toth found at https://www.youtube.com/playlist?list=PL1P11yPQAo7pH9SWZtWdmmLumbp_r19Hs.