Dinosaur Tactic

Detail Design Document

I. Introduction

An endless running game written in python with a pyxel framework that features the player as a running caveman trying to collect food while avoiding dinosaurs on his way.

II. Gameplay

The player is chased by a hidden dinosaur (from the left side of the screen) and must run constantly. Points are earned by collecting food and running from the dinosaurs.

The player must avoid dinosaurs coming up from the right screen side by moving up and down. Moreover, the player has a booster system that can thrust the caveman to the right, which is tracked by a booster meter beneath the caveman.

III. Game Logic

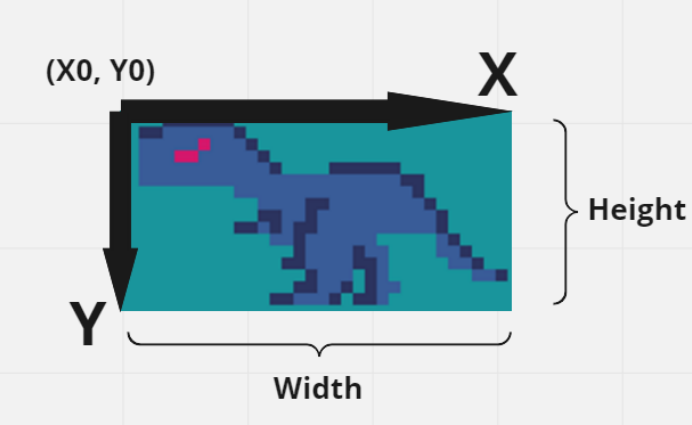
The pyxel framework provides a useful model to compile three main functions: “init”, “update”, and “draw”. In order to utilize this framework, it is better to design every entity in the game as an object with three similar functions as well. Therefore, every entity in the game – player, dinosaur, and food all have the same layout of those functions. However, each entity can be added to or removed from properties that can be adjusted to customize the entity’s behaviors.

Pyxel provides the “App()” class (the model discussed above) that acts like a game engine that can render and update, responsible for setting up the program, receiving inputs while drawing, and updating all entities of each frame.

Pyxel framework at its default will render 60 frames per second. This will be the timing system we can use by providing the developer with the ability to get current frame counts since the program started.

Coordinate:

The canvas that the App() class draws upon has a coordinate system, whose origin is at the top left corner. Each entity also has its own coordinate configuration to be drawn.



Animation:

With the ability to get frame counts, animation can be created by drawing consecutive images of an entity, and each image takes up a short period. The effect can be implemented by using mod calculations to set up different periods for each image.

```
if(pyxel.frame_count % 20 > 7):
  pyxel.blt(self.x, self.y, 0, 16, 0, self.w, self.h)
else:
  pyxel.blt(self.x, self.y, 0, 16, 8, self.w, self.h)
```

Collision:

The game's only interaction between different entities is collisions. This happens when the box border of an entity overlaps with another. This can be implemented easily by checking entities' coordinates, height, and width. This is an example of a check between a dinosaur and the player's box borders:

```
self.player.x + self.player.w > dino.x
and dino.x + dino.w > self.player.x
and self.player.y + self.player.h > dino.y
and dino.y + dino.h > self.player.y
```

IV. System design

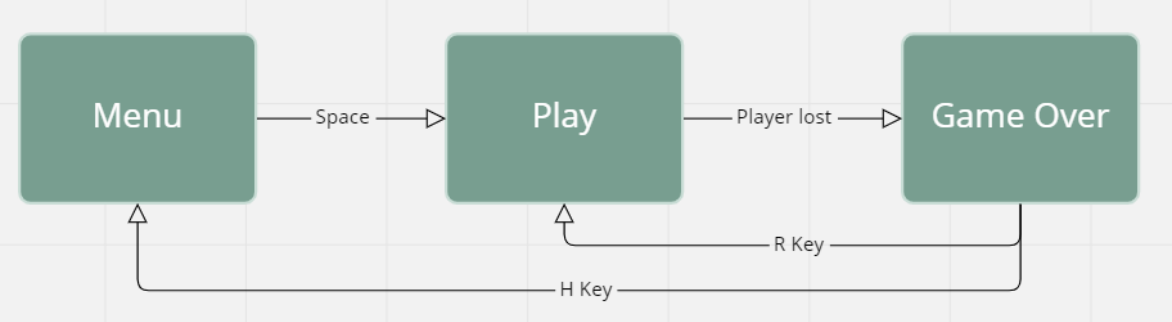
1. Screen System & Game Flow

The flow of the game can be represented by three different scenes: SCREEN_MENU (the menu screen to display pre-game instructions & the highest score), SCREEN_PLAY (the screen to play), and SCREEN_GAMEOVER (the screen to display the game over state and present player with last game score vs high score). A property in the "App()" class called "game_state" is used to control which screen is being rendered.

The enum class is created, which is called "Game_State", to better develop the game by using the enum and match case syntax.

The update and draw functions of the App() class are also divided to three screens: update_screen_play(), update_screen_menu(), update_screen_over(), draw_screen_play(), draw_screen_menu(), draw_screen_over(). With this design, more screens can be added in the future with ease if needed (Ex: shop screen, inventory screen…)

Screens can be presented in the diagram below and can be controlled by player input.



2. Configuration & Utilities

- Screen configuration used:

  SCREEN_WIDTH = 240
  SCREEN_HEIGHT = 150

- Game configuration used:

  Player's spawn coordinates:
  PLAYER_X_START = 10
  PLAYER_Y_START = SCREEN_HEIGHT/2 - 8
  Game speed and acceleration:
  GAME_SPEED_NORMAL = 0.5
  GAME_ACCELERATE_NORMAL = 0.001
  Global variables:
  high_score = 0
  current_game_speed = GAME_SPEED_NORMAL
  current_game_acc = GAME_ACCELERATE_NORMAL

Since there are various dinosaur and food entities in this game, controlling those entities by using arrays to store each kind will produce much more optimized and clean code. The assisting

function is key to implementing this design. The first function is “def update_list(list)”, which will update every element inside the list. The next function is “def draw_list(list)”, which will draw every element of a list. The last one, “def cleanup_list(list)”, will remove all the elements that are not in use in the game. This can be accomplished by marking every entity with “is_alive” property, which will be set to “False” whenever that entity is not used. The function “cleanup_list(list)” will then go in the array and remove all elements that are marked as not alive (is_alive = False). This makes the game run faster and more optimized by removing all unused entities that can accumulate into a large memory. Afterward, the array of each kind of entity is set and ready to use.

```
blasts = []
enemy = []
food = []
```

There exists another helper function called “reset_game_setting()” that helps reset all global variables whenever the player starts a new game.

3. Music System

All music sound setups can be found within the function “sound_set_up()”. In this function, two sound effects (1st one plays when the player collects food, 2nd will play when the player dies and the blast appears) and a music track are set up to be used in the framework. This function will be called in the “init” function of the “App()” class.

The music sound can be turned on and off by player input in the menu screen (the information will be displayed as the sound icon and crossed sound icon), which is achieved by setting a music property in the “App()” class (True is on, False is off). The music control key is the M key. The music will be reset whenever the player started a new game while the sound effect will always be played in each game.

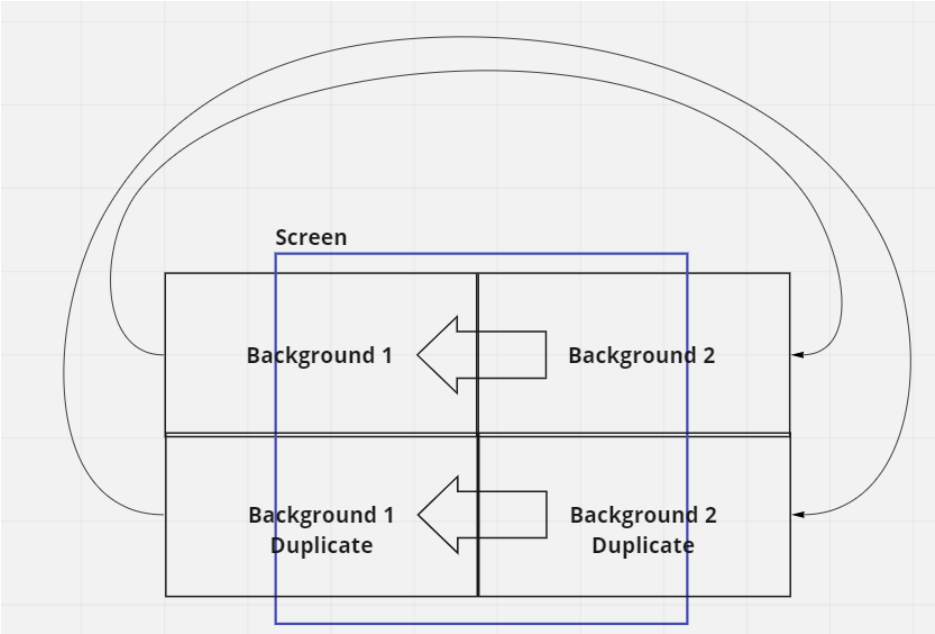
4. Endless Background

The background itself is an entity that is represented by the Background class. The background class has two coordinate data (x1, y1) and (x2, y2). This is used to create an endless running background effect by seamlessly moving 1st background toward the left side while the 2nd background appends to the end of the 1st background (also moving to the left with the same speed). When the 1st background moves to the very left and is hidden, it then appends to the end of the 2nd background and this process will happen forever.

```
def update(self):
        self.x1 -= current_game_speed
        self.x2 -= current_game_speed
        if(self.x2 < 0):
```

```
            self.x1 = self.x2 + 256
        if(self.x1 < 0):
            self.x2 = self.x1 + 256
```

If the SCREEN_HEIGHT is larger than the background image's height, this process can be implemented vertically by drawing two duplicate backgrounds that mimic the 1st and 2nd backgrounds' behavior underneath.



5. App Class

The "App()" class stands as a gateway to control which screen is being displayed on the canvas with the help of the "game_state" property. This class also stores all variables and properties that are needed to enhance the player's experience: music, current score, or last score….

Property

```
        self.background = Background()
        self.last_score = 0
        self.music = True
        self.player = Player(PLAYER_X_START, PLAYER_Y_START)

        self.current_score = 0
```

Reset

This function is used to reset all the game global variables needed to start a new game.

Screens

Menu Screen

Update: Looks for inputs from the keyboard to change among screens or mute music

Draw: Draws title, introduction, high score, background, and music control

Game Screen

Update: Updates the game, spawns dinosaurs, foods, blasts, player, and background

Draw: draws all game entities and the current score

Over Screen

Update: Looks for inputs from the keyboard to replay or go back to the home screen

Draw: Shows last score, high score, and text instructions

6. Points

Points in the game are earned in three different ways:

- The player earns points for every second he keeps the game running
- The player earns more bonus points when the game speed he plays at becomes higher
- The player earns points when collecting food

After losing, the new high score will be set if the current score is higher than the current high score. The high score is not stored within the program, but only when the program runs.

```
if(int(pyxel.frame_count / current_game_speed) % 60 == 0 or pyxel.frame_count % 60 == 0):
            self.current_score += 1
```

7. Enemies System (Dinosaur)

There are multiple dinosaurs that the player needs to avoid when playing. All of them appear on the right side of the screen and then gradually move to the left side. If a dinosaur doesn't collide with the player, it is removed after disappearing from the screen. There are three types of dinosaurs with their own unique properties and behaviors. There is a limit to the maximum number of dinosaurs rendered in the game, which is 30. Every dinosaur spawned in the game will automatically append itself to the enemy array. All dinosaurs have an "animation(self)" function to draw animation.

Small Rex

Appearance: every 25 frames, on the right of the screen with a random height

Property: a small dinosaur that moves fast (8x8)

Behavior: moves towards the left side

Big Rex

Appearance: every 80 frames, on the right of the screen with a random height

Property: bigger than small rex, lower speed (16x32)

Behavior: moves towards the left side

Flying Dinosaur

Appearance: in packs, every 120 frames, on the right of the screen with random height

Property: has acceleration, high speed

Behavior: has a warning display for three seconds, then move towards the left side

8. Food System

Food is the entity that the player can collect to have more points. Each food collected can give the player 10 points and create a sound effect. There is a maximum number of food entities that can be in the game, which is 30.

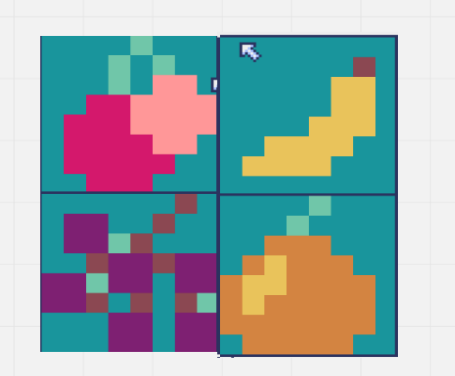
Appearance

A food entity is created every 30 frames, or around 0.5 seconds. A random integer will be generated to choose the food type. The food is placed at a random height on the right side of the screen and will move to the left. Any food created is automatically appended to the food list. If the food is not collected after moving to the very left side of the screen, it will be removed.

```
r = random.randint(0, 3)
if(pyxel.frame_count % 30 == 0 and len(food) < 30):
            new_y = random.randint(0, SCREEN_HEIGHT - 8)
            Food(SCREEN_WIDTH - 8, new_y, r)
```

Types

There are four different types of food, which can be customized by sending input when creating the object as type = int from 0 to 3 (default is 0).



9. Blasts

The blast class represents the explosion entity, which is created every time the player collides with a dinosaur. The blast array is used instead of the blast property in the “App()” class because the player can collide with more than one dinosaur upon death. This situation will generate more blasts to create a dramatic effect when losing the game.

The blast has a default configuration:

BLAST_START_RADIUS = 1
BLAST_END_RADIUS = 8
BLAST_COLOR_IN = 7
BLAST_COLOR_OUT = 10

As the blast appears, it has a starting radius, then expands to the ending radius, and finally ends its explosion. The game switches to the game-over screen only when all the blasts end.

```
if(len(blasts) == 0 and not self.player.is_alive):
            self.game_state = Game_State.SCREEN_GAMEOVER
```

10. Player

The caveman is a special entity that can be controlled by the player's inputs.

Appearance

The player appears in the middle of the SCREEN_HEIGHT and on the left side of the screen. The appearance coordinates can be changed in the configuration on "PLAYER_X_START" & "PLAYER_Y_START".

Animation

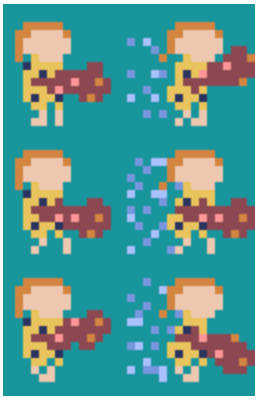
The caveman has two types of animation, including running animation and speeding animation. The speeding animation is displayed when the player presses the D key and still has some booster left. Two types of animation correspond to two animation functions that are called in the player's "draw" function.

Control

The player can make the caveman move up and down by the W and S key with no restriction, except for moving to the outside of the screen. The player also has a booster system that can thrust the player to move towards the right side in a short amount of time, which can be activated by the D key.

Booster

The player is allowed to move the caveman faster to the right side of the screen in a short amount. It is designed for the caveman to never exceed 33.33% of the screen width. It takes approximately 142 frames, or 2.3 seconds for the player to refill the booster meter. If the player ran out of his booster, he cannot use it until it is refilled. Also, if the player's current x-position is larger than the player's x-starting-position, the player will be dragged back to his original position of "PLAYER_X_START".



(Note: Further explanation of the code is written as comments inside the main.py file)