Hungry Duck

Game Design Document

Sabrina Mahoney

## Overview

"Hungry Duck" is a 2D arcade game using pygame and simpleGE.

The game: The player is a duck. The duck is at the bottom of the screen with the image of water behind it. The user can move the duck from left to right and up and down with the arrow keys. A few pieces of bread fall from the top of the screen. The bread will fall from a different x positions, and at a different speed between 3 and 8 pixels per frame down the screen. If the duck touches the bread, a positive sound effect is played, and the player's score goes up. If the bread falls past the bottom of the screen, it is reset to a new random position at the top of the screen and a new falling speed. There will also be 3 logs falling from the top of the screen. If the duck touches a log a negative sound effect will play and they player will lose a point. The game continues for a set period of time (20 seconds for playtesting purposes)

When the game begins, it will show an intro screen with instructions and two buttons. The play button will take the game into the play state. The quit button will exit the game.

After the player has played a round of the game, they are taken back to the intro screen. This will show the latest score.

## State Transition Diagram

A diagram of a game

Description automatically generated with medium confidence

The user will start on the intro screen. If the user chooses to play the game, they are sent to the game play scene. If they choose to quit, the game ends.

## If the player runs out of time they will also be sent back to the intro screen.

## The Instructions Scene

A paper with writing on it

Description automatically generated

This scene has four main visual elements:

* **instructions** - a stock simpleGE multiLabel containing instructions for game play
* **prevScore** - a stock label showing the previous score
* **btnPlay** - a stock button indicating "Play"
* **btnQuit** - a stock button indicating "Quit"

Other attributes:

* **prevScore** - integer indicating last score, passed into the class initializer and displayed on prevScore label
* **response** - string representing the user's intentions. Set by the two buttons and read in the main function

Initializer will create all attributes and set up the sprite list

init(score):

Set image to blanket.png

Set response to "Play"

Create instructions MultiLabel

Add textLines containing instructions

Set instructions center to (320, 240)

Set instructions size to (500, 250)

Copy score parameter to prevScore attribute

Create LblScore

Set text to "Last score: {prevScore}"

Set center to (320, 400)

Create btnPlay

Set text to "Play"

Set center to (100, 400)

Create btnQuit

Set text to "Quit"

Set center to (550, 400)

Add lblInstructions, lblScore, btnQuit, and btnPlay to sprites

All event-handling will happen in the scene's process() method

process():

If the quit button is pressed:

Set response to "Quit"

Stop the scene

If the play button is pressed:

Set the response to "Play"

Stop the scene

## 

## The Game class

A piece of paper with writing on it

Description automatically generated

(background water)

The Game class will have a number of visual attributes:

* **duck**- an instance of the **duck** class (see below)
* **bread** - a list of instances of the **bread** class (see below)
* **log**- an instance of the **log** class (see below)
* **lblScore** - an instance of the **LblScore** class (see below)
* **lblTime** - an instance of the **LblTime** class (see below)

It will also contain some non-sprite assets:

* **timer** - a stock instance of the simpleGE.Timer class
* **score** - an int containing the current score
* **sndBread**- a stock instance of the simpleGE.Sound class
* **sndLog**- a stock instance of the simpleGE.Sound class

Initializer will create all the needed components:

init:

Set image to water.png

Create timer

Set timer's total time to 10 (for now)

Set score to zero

Initialize sndBread to bread sound effect

Initialize sndLog to log sound effect

Create instance of duck -> duck

Create list of (5) Bread instances -> bread

Create list of (3) Log instances -> log

Create instance of LblScore

Create instance of LblTime

Add duck, bread, log, lblScore, lblTime to sprites

All event-handling will occur in the scene's process() method:

process:

For each bread in the bread list:

If that bread collides with duck:

Play the bread collision sound (sndBread)

Reset that bread

Add one to the score

Update lblScore to indicate the new score

For each log in the log list:

If that log collides with duck:

Play the log collision sound (sndLog)

Reset that log

Subtract one to the score

Update lblScore to indicate the new score

Update lblTimer with the current time left

If the time left is less than zero:

(for now) print the score to the console

Stop the game

## 

## Components of the Game class

### Duck

Duck is a subClass of simpleGE.Sprite

The image should be a fair-use version

Size should be roughly 50 by 50

Transparent background is preferred

Initial position center bottom of screen

moveSpeed attribute is an integer starts at 5

init:

Set image to duck.png

Set size to 50x50

Set position to (320, 400)

Set moveSpeed to 5

All event-handling will be in process() method

Move left on left key, right on right key

Move up or down

process:

If left key is pressed

Subtract moveSpeed from x

If right key is pressed

Add moveSpeed to x

If up key is pressed

Add moveSpeed to y

If down key is pressed

Subtract moveSpeed from y

### Bread

Bread is a subclass of simpleGE.Sprite

The image should be a free image of bread

It should have a transparent background

Reset method sets coin to top of screen, random position

Fall speed is random within limits (3 to 8 ppf for now)

Bread falls down screen

If Bread leaves bottom of screen, reset

Bread-duck collision handled at game level, not needed here

Bread has no special attributes, but three methods

* **init()** - standard initialization
* **reset()** - custom method to set speed and position
* **checkBounds()** - overwrite existing checkBounds to handle bottom-of screen

init():

Set image to bread.png

Set size to 25x25

Call reset()

reset():

Set y to 5

Set x to random from zero to screen width

Set dy to random between 3 and 8

checkBounds():

If bottom of sprite is larger than screen width:

Call reset()

### Log

Log is a subclass of simpleGE.Sprite

The image should be a free image of a log

It should have a transparent background

Reset method sets coin to top of screen, random position

Fall speed is random within limits (3 to 8 ppf for now)

Log falls down screen

If Log leaves bottom of screen, reset

Log-duck collision handled at game level, not needed here

Log has no special attributes, but three methods

* **init()** - standard initialization
* **reset()** - custom method to set speed and position
* **checkBounds()** - overwrite existing checkBounds to handle bottom-of screen

init():

Set image to log.png

Set size to 25x25

Call reset()

reset():

Set y to 3

Set x to random from zero to screen width

Set dy to random between 3 and 8

checkBounds():

If bottom of sprite is larger than screen width:

Call reset()

### LblScore

LblScore is a subclass of the simpleGE.Label

It is quite simple - could have been a stock instance

It simply has text and center, no events

init():

Set text to "Score: 0"

Set center to (100, 30)

### LblTime

LblTime is also a simple subclass of simpleGE.Label

Again, only text and center, no events

init():

Set text to "TimeLeft: 10"

Set center to (500, 30)

## The main() function

The main function will manage the high-level state transition between intro and play states.

It is a very standard main loop, containing four variables:

* **instructions** - an instance of the Instructions class
* **game** - an instance of the Game class
* **keepGoing** - the classic Boolean sentry
* **score** - the current score

Psuedocode for main

main():

Set keepGoing to true

Set score to zero

While keepGoing is true:

Create an instance of Instructions -> instructions

Pass the current score to instructions as a parameter

Start instructions

When instructions ends,

If instructions.response is "Play":

Create an instance of Game -> game

Start game

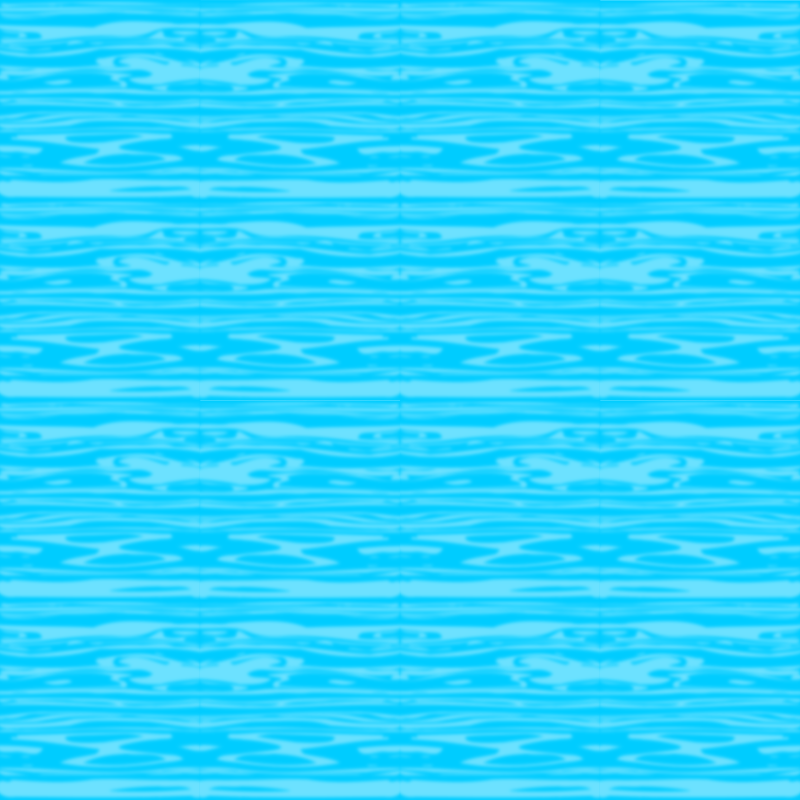
When game is over, copy game.score to score

If instructions.response is anything but "Play":

Set keepGoing to False, which will exit the game

## Asset plan

**Water.png**



Fair use from openclipart.org

**Duck.png**

A pixel art of a duck

Description automatically generated

Fair Use from openclipart.org

**Bread.png**

A piece of bread with a black background

Description automatically generated

**Fair use from openclipart.org**

**Log.png**

A brown and black wood

Description automatically generated with medium confidence

**Fair use from openclipart.org**