Catch the Cash

Game Design Document

Sam Allen

Overview:

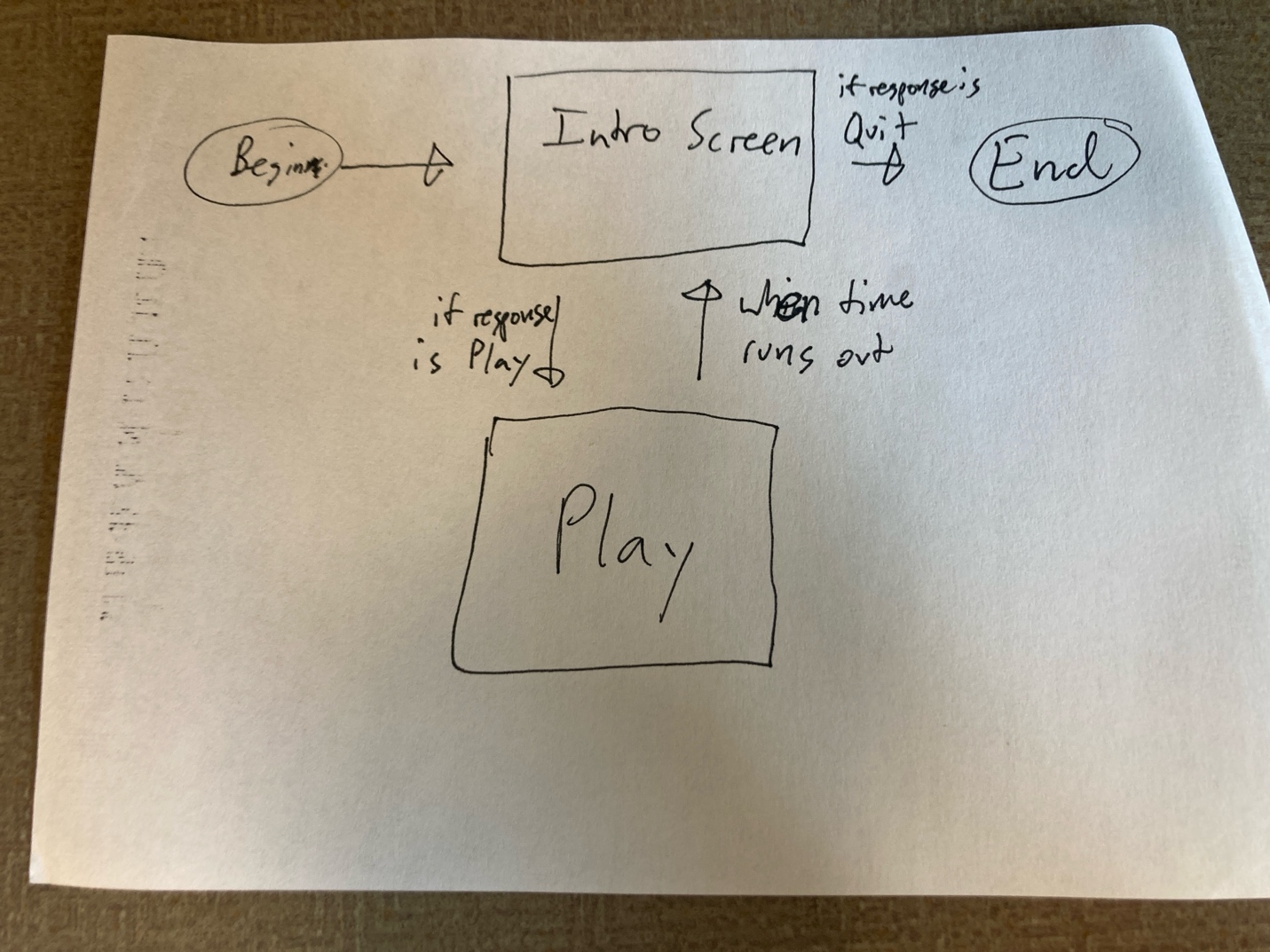
Catch the Cash will use a basic 2D arcade game set up to demonstrate how pygame and simpleGE work.

The player is a pink and purple felted creature called Ronald. It appears at the bottom middle of the screen with an image of mountains in the background. The player can move Ronald left and right with the arrow keys. Mushrooms will begin to fall straight down from the top of the screen at different x positions and speeds ranging from 3 to 8 pixels/frame. If Ronald touches a mushroom an eating sound will play, the mushroom will reappear at the top of the screen, and the player will get a point. If a mushroom leaves the bottom of the screen, it will reappear at a random x position at the top of the screen with a random falling speed between 3 and 8 pixels/frame. The game goes on for a period of time until it will end (10 seconds for testing purposes).

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

After the player completes a round of the game, they are taken back to the intro screen, which will display their last score.

**State Transition Diagram**

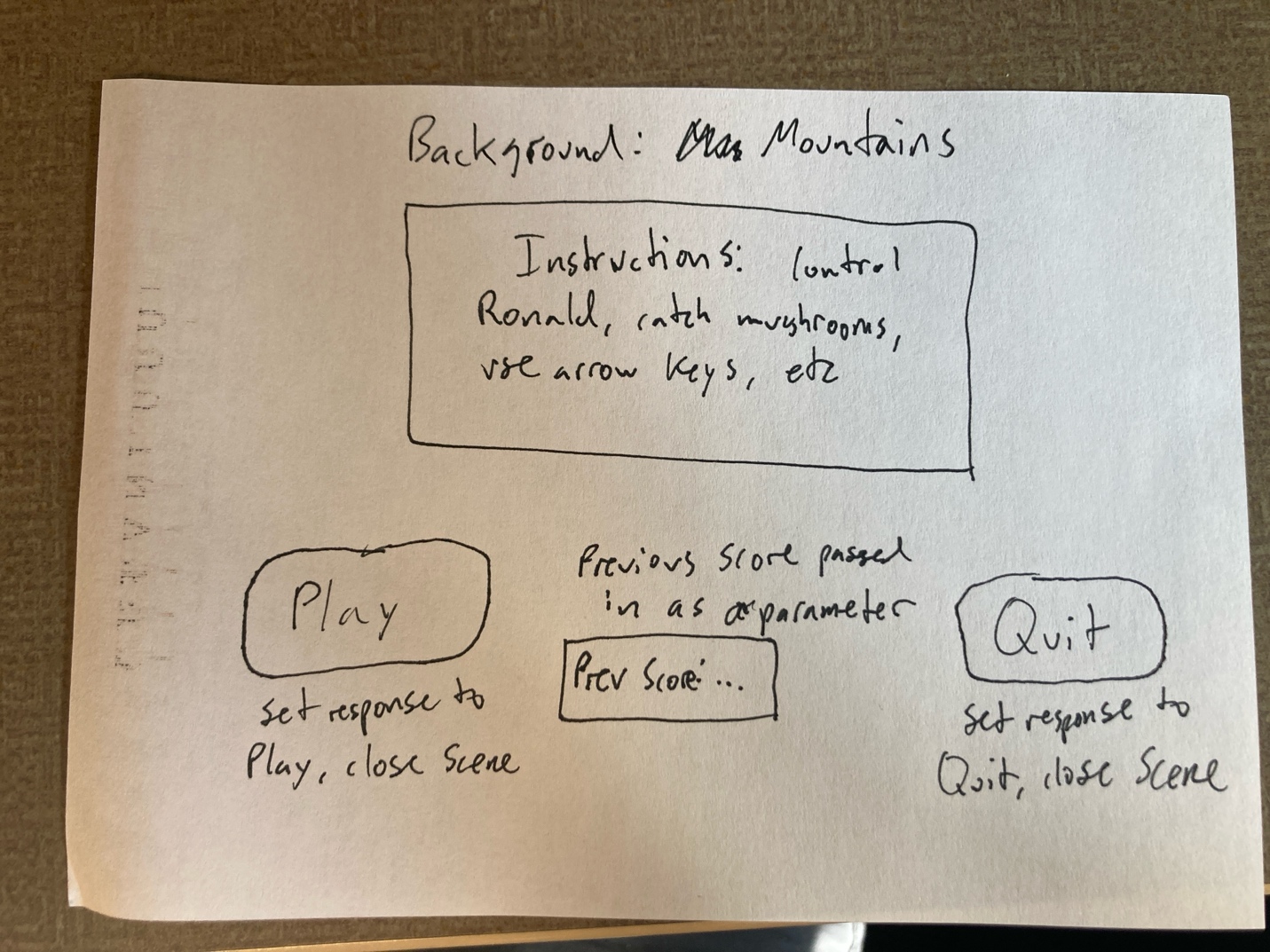


The game will have a two-state system. Each state will be represented by a subclass of the simpleGE Scene class. The player is first sent to the Intro Screen Scene. This will have instructions displayed along with 2 buttons. Both buttons close the scene but will set a response variable indicating the user’s choice. If they chose to play the game, they are sent to the play Scene. If they chose to quit, the game will end.

The game play scene always ends when the time expires and returns the player to the intro scene. It also passes the score back to the main function, which will be displayed in the intro scene.

**The Instructions Scene**

Simple screen that controls access to the game



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”, leading to the play state
* btnQuit: a stock button indicating “Quit”, exiting the program

Other attributes:

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

Initializer will create all the attributes and set up the sprite list

Init(score):

Set image to mountainBackground.jpg

Set response to “Play”

Create instructions MultiLabel

Add tetLines containing instructions

Set instructions center to (320, 250)

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)

Add lblInstructions, lblScore, btnQuit, and btnPlay to sprites

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

Process():

If the quit button is pressed:

Set response to “Quit”

Stop the scene

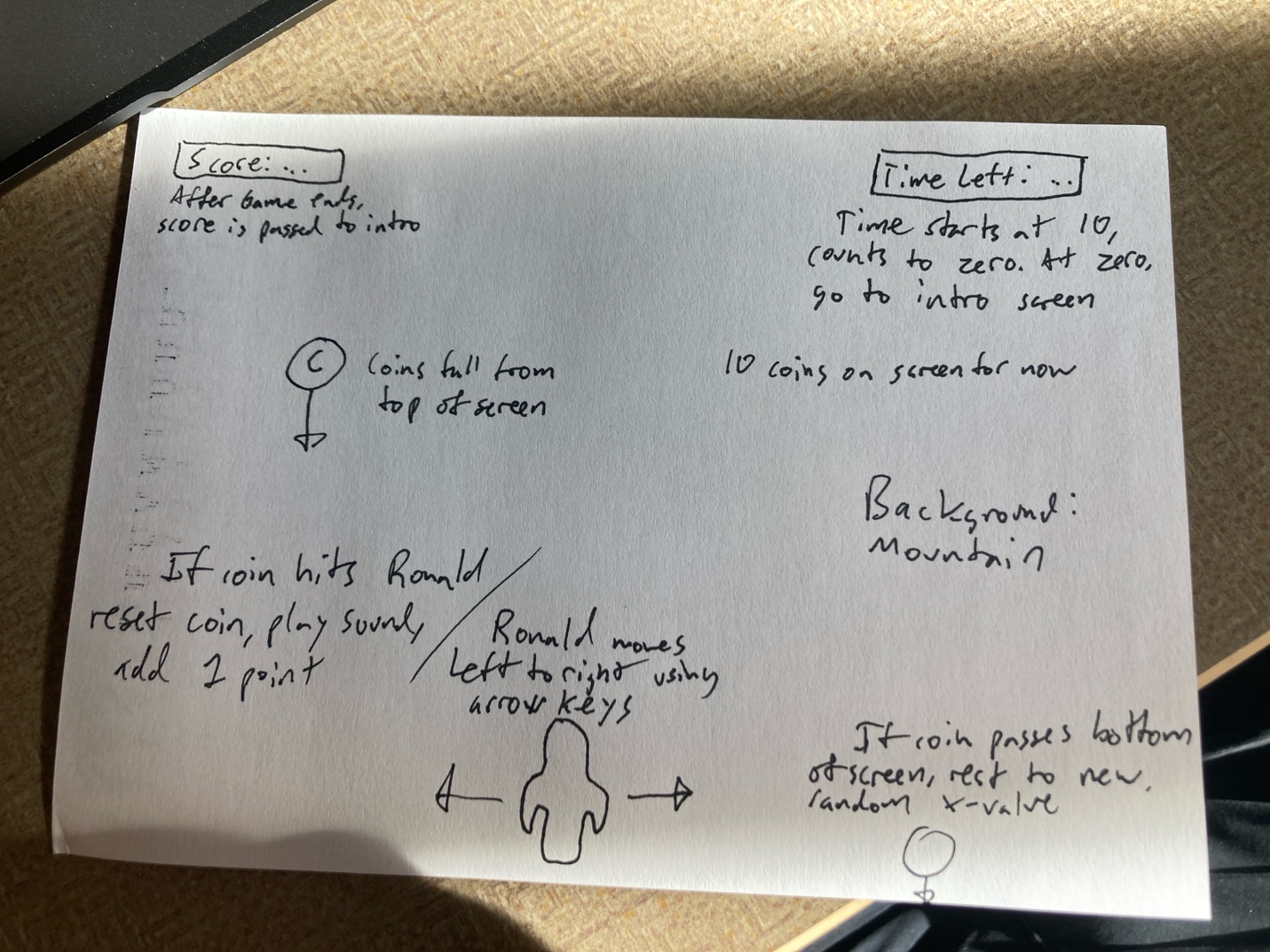
If the play button is pressed:

Set the response to “play”

Stop the scene

**The Game Class**

This is the primary class of the game, it will be subclassed from simpleGE.Scene



Game class will have a number of visual attributes

* Ronald: made from an instance of the Ronald class (see below)
* Mushrooms: a list of instances of the Mushroom 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 integer variable containing the current score
* sndMushroom: a stock instance of the simpleGE.Sound class

Initializer will create all the needed components

Init(game):

Set image to mountainBackground.jpg

Create timer

Set timer’s total time to 10

Set score to zero

Initialize sndMushroom to mushroom sound effect

Create instance of Ronald 🡪 Ronald

Create list of 10 Mushroom instances🡪 mushrooms

Create instance of LblScore

Create instance of LblTime

Add Ronald, mushrooms, lblScore, lblTime to sprites

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

Process:

For each mushroom in the mushrooms list:

If that mushroom collides with Ronald:

Play the mushroom collision sound (sndMushroom)

Reset that mushroom

Add 1 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 testing) Print the score to the console

Stsop the game

**Components of the game class**

Each of the visual elements of the Game class is an extension of a simpleGE element

Ronald

Ronald is a subclass of simpleGE.Sprite

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(Ronald):

set image to feltRonald.png

set size to roughly 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

Process:

If left key is pressed:

Subtract movespeed from x

If right key is pressed:

Add movesSpeed to x

Mushroom

Mushroom is a subclass of simpleGE.Sprite

Should have a transparent background

Reset method sets coin to the top of the screen, random x position

Fall speed is random within 3-8 (for now)

Mushroom falls down screen

If mushroom leaves the bottom of the screen, reset

Mushroom-Ronald collision handled at game level, not here

Mushroom 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 feltMushroom.png

Set size to 25x25

Call reset()

Reset():

Set y to 10

Set x to random from zero to screen witch

Set dy to random between 3 and 8

checkBounds():

If bottom of sprite is larger than screen height:

Call reset()

**LblScore**

LblScore is a subclass of the simpleGE.Label

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

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 class Boolean sentry
* score: the current score

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

Notes on the main loop

* For technical reasons, it’s best to re-create each of the scenes on each pass
  + It’s inefficient but easier to maintain
* Each scene’s start() method is *blocking*
  + Which means the next line of main will not occur until the scene is closed

**Milestone Plan**

General strategy is to create gameplay first, then instructions screen, and finally integrate with state management. Game process will be stored on github, with a marked commit for each milestone reached and multiple other commits as needed. Each milestone commit will run correctly with the milestone demonstrated. Each milestone is expected to take on programming session to complete.

1. Game scene with background image
2. Add basic Ronald sprite
3. Add keyboard motion to Ronald
4. Add single coin with reset, falling, and boundary behaviors
5. Add collision effect between Ronald and mushroom, sound effect
6. Modify for multiple (10) coins including collision behavior
7. Add scorekeeping, timing, and appropriate labels
8. Add instructions class and state transition

**Asset plan**

mountainBackground.jpg

A landscape with a mountain and clouds in the sky

Description automatically generated

I took this photo

feltRonald.png

A stuffed toy with legs and arms

Description automatically generated

I made this creature, took the photo, and edited it

feltMushroom.png

A stuffed mushroom with a white circle

Description automatically generated

I made this creature, took the photo, and edited it

Mushroom.wav

Custom audio by Sam Allen created with jsfxr: <https://sfxr.me/>