Catch the Cash

Game Design Document

Andy Harris

## Overview

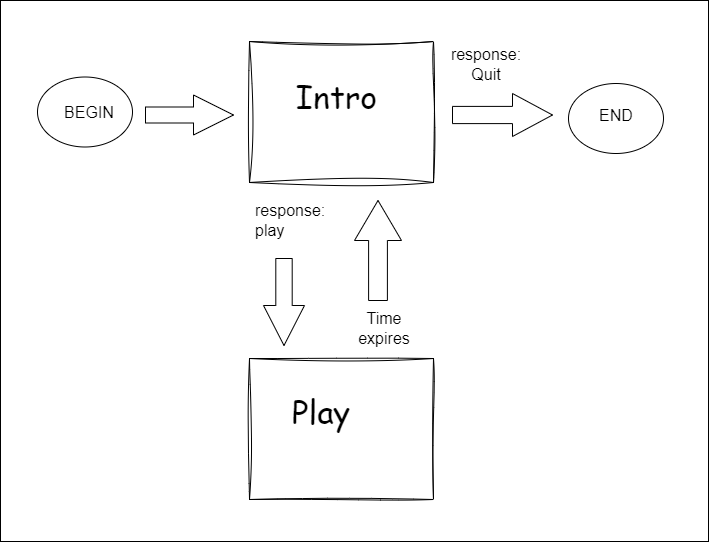
"Catch the Cash" will be a basic 2D arcade game to demonstrate the overall flow of a game using pygame and simpleGE.

The premise is extremely simple: The player is Charlie the Cardinal, the BSU mascot. Charlie appears near the bottom of the gameplay screen with a background image of the BSU campus behind him. The user can move Charlie to the left and right with the corresponding arrow keys. A series of coins fall from the top of the screen. Each coin will fall from a different x position, and at a different speed between 3 and 8 pixels per frame straight down. If Charlie touches a coin, a positive sound effect is played, and the player's score is increased. If a coin leaves the bottom of the screen, it is reset to a new random position at the top of the screen and a new falling speed. The game continues for a set period of time (ten 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, she is taken back to the intro screen. This will display the latest score, motivating the player to play again.

## State Transition Diagram



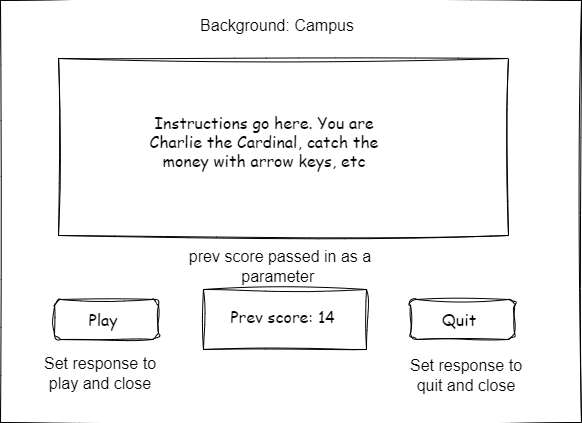
This game will have a very standard two-state system. Each state will be represented by a subclass of the simpleGE Scene class. The player is initially sent to the Intro scene. This will display instructions and two buttons. The buttons both close the scene, but before closing the scene, they will set a response variable indicating the user's preference. If the user chooses to play the game, she is sent to the game play scene. If she chooses to quit, the game ends.

The game play scene always ends when the time expires, and always returns control to the intro scene. However, it does pass back its score to the main function, which uses that score to provide feedback to the user in the intro scene.

## 

## The Instructions Scene

The instructions scene is simple but vital: It 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"
* **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 campus.jpg

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:

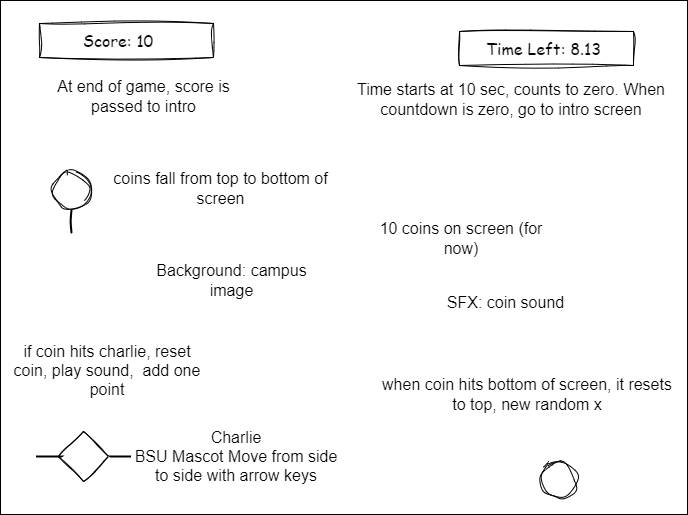
Set the response to "Play"

Stop the scene

## 

## The Game class

This is the primary class of this game. It will be subclassed from simpleGE.Scene



The Game class will have a number of visual attributes:

* **charlie** - an instance of the **Charlie** class (see below)
* **coins** - a list of instances of the **Coin** 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
* **sndCoin** - a stock instance of the simpleGE.Sound class

Initializer will create all the needed components:

init:

Set image to campus.jpg

Create timer

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

Set score to zero

Initialize sndCoin to coin sound effect

Create instance of Charlie -> charlie

Create list of (10) Coin instances -> coins

Create instance of LblScore

Create instance of LblTime

Add charlie, coins, lblScore, lblTime to sprites

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

process:

For each coin in the coins list:

If that coin collides with charlie:

Play the coin collision sound (sndCoin)

Reset that coin

Add 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

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

### Charlie

Charlie is a subClass of simpleGE.Sprite

The image should be a fair-use version of the BSU mascot

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 charlie.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

process:

If left key is pressed

Subtract moveSpeed from x

If right key is pressed

Add moveSpeed to x

### Coin

Coin is a subclass of simpleGE.Sprite

The image should be an encumbrance-free image of a coin

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)

Coin falls down screen

If coin leaves bottom of screen, reset

Coin-charlie collision handled at game level, not needed here

Coin 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 coin.png

Set size to 25x25

Call reset()

reset():

Set y to 10

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

Notes on the main loop

* For technical reasons, it's best to re-create each of the scenes on each pass
* (it's a bit inefficient, but much 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 one programming session to complete.

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

## Asset plan

**Campus.jpg**



Fair use from BSU.edu

**Charlie.png**



Fair Use from BSU.edu

**Coin.png**



Creative commons:

<https://opengameart.org/content/coin-icon>

**Coin.wav**

Custom audio by Andy Harris created with jsfxr: <https://sfxr.me/>