Developed and designed by Christo Savalas

Introduction

Whack-a-Mole is a classic arcade game favorite loved by man and loathed by moles. In *Whack-a-Mole: The Reckoning*, the player finds themself in a post-apocalyptic world brought about by disgruntled, nihilistic rodents with nothing to lose. Indeed--and unlike the arcade classic--the player will not only need to whack the moles, but also be on the lookout for the extra aggressive critters who fight dirty, as well as pacifists who just want to be left alone.

Gameplay

Upon opening the game, the player is met with a disturbing view of a once beautiful city, now overrun by vigilante moles, who the player can see on all corners of the screen just waiting for a taste! The player presses start to enter into one of four random mole fields:

Desert

Nuclear Wasteland

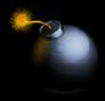
Hell

Antarctica

The player now has but 60 seconds to vanquish as many of the (admittedly well-dressed) moles as possible. Luckily, the player is not empty handed. A trusty mallet, one for each locale, appears on screen ready to start whacking!

Unlike the arcade game, these moles sometimes have devastating items that they use to fight back! These items include:

Bomb



If the player doesn't hit a mole with this bomb in time, it will blow up the whole mole field, ending the game immediately. The player is rewarded extra points for hitting moles with these.

Tomato

Perhaps not as fear-inspiring as the bomb, a mole with a tomato will throw the fruit at the player's screen, cracking it and knocking points off the player's score.

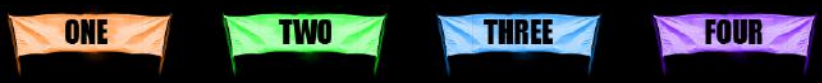
Money Bag

Moles with money are a welcome sight. Hitting these moles will increase the player's score.

White Flag

Aside from the moral crisis the player ought to experience after hitting a white flag-carrying mole, the player's score will also drop precipitously. These moles just want peace.

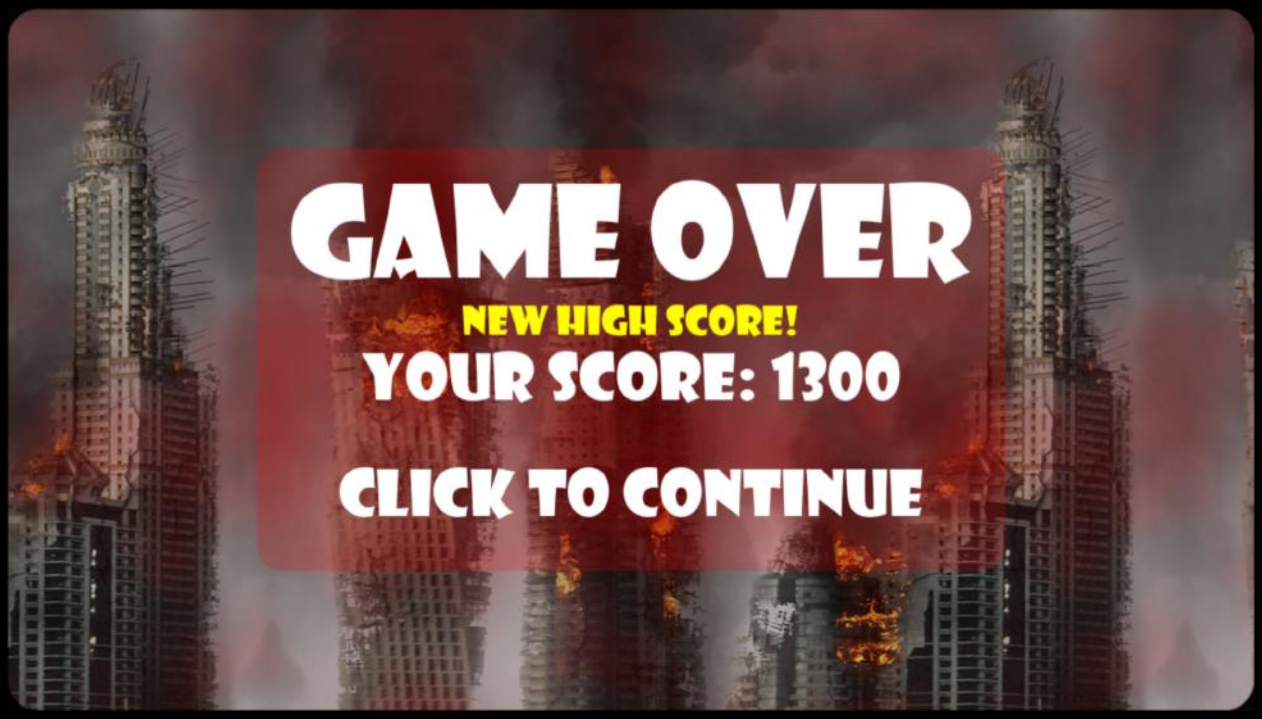
Ordered Banners



Banner-wielding moles must be hit in ascending order, otherwise they will all disappear and the player's score will be penalized. If successful, the score reward is substantial.

The player must bravely battle in this arena for a full minute of glorious combat:

Once the 60 seconds are almost up, the player hears an audible warning tone, alerting the imminent conclusion of the adventure. Once over, the player is presented with their score, and whether it was a new high score:



“Only the dauntless will bear fruit. The pigeonhearted will know no relief.” -Anonymous Mole

Installation Instructions

1. Download WAM-TheReckoning.zip
2. Unzip WAM-TheReckoning.zip to the desired directory
    a. This will create a folder called WAM-TheReckoning
3. Drag and drop the WAM-TheReckoning folder onto your love2d executable
4. Enjoy 👍🏽

Alternative Installation Instructions

1. Rename WAM-TheReckoning.zip to WAM-TheReckoning.love
2. Simply double click WAM-TheReckoning.love to open the game
3. Enjoy 👍🏽

Input Options

- Mouse
    - Interaction with UI items and gameplay
- Touchscreen
    - Interaction with UI items and gameplay
- Keyboard
    - Interaction with UI items

Technical Background

I programmed *Whack-A-Mole: The Reckoning* in the Love2D framework and used code from the project *Match3* written by Colton Ogden as a base template on which to build out the code. This is noted in the comments of the source files. I chose to abstract functionality of the game into discrete classes in order to ensure extensibility and ease of reading by third parties. However, I tried to limit this to functionality that lent itself to abstraction.

Wherever there were redundancies (repeated code), I split it off into a separate function that could be called from different areas of the codebase. I also ensured to use constants where appropriate, to avoid having to change values in multiple areas of the codebase. I commented the document very heavily should I ever want to continue developing this game, and so you would be able to quickly know exactly what any particular area of the code is doing.

Lastly, the game was originally intentionally low-resolution, with the nearest neighbor filter to give it a retro look. However, I had made nice looking, high res assets, so I decided to modify the game to run in high res. This process was long, mostly due to all of the static numbers from the original Match3 distro code, which only worked for that one resolution. So, as I went, I was sure to declare all sizes and positions in relative terms proportional to the screen resolution. That way, the game could be quite easily modified to run at different resolutions in the future, with minimal effort.

main.lua

Much of the code in here is boilerplate, however, there are some key additions:

- Loading of a previous high score from the file system, if any exists
- Addition of functions that keep track of whether the mouse was moved
- Foreground cloud effects with random lightning strikes
    - Lightning effect achieved by tinting the clouds temporarily

util.lua

This file contains 4 utility functions:

- A function to generate quads from the sprite sheet
    - The sprite sheet contains sprites of varying sizes, and orientational grouping. I made a single function to handle all of these cases to cut down on clutter.
- Two functions which work in tandem to create the effect of text being typed to the screen
- A function to print tables, sourced from coronalabs, very useful for debugging

LoadingState.lua

This file contains the logic and display of the loading state (Note: the game doesn't actually need to load, it is just to display an intro credits message, in this case: "Presented by Christo Savalas")

- The intro message is typed onto the screen, and when complete, is animated with a little blip sound effect
- It also starts the game music looping
- While this functionality is relatively small, and could have been done in main.lua, I wanted to avoid cluttering it up unnecessarily, and it is certainly a discrete game state, conceptually.

StartState.lua

This file contains the logic and display of the start state (right after the loading screen or after a game over state. Notable features:

- Since the game is all mouse based, I added mouse functionality to the start page, as well as allow keyboard input
    - This also allows the game to be played on a tablet, which is quite fun.
- The title is animated on game open to lend some drama
- To further add to the drama on game open, a massive explosion effect with a sound effect is shown (apologies to headphone users)
- Moles randomly pop out from all areas of the screen as well

BeginGameState.lua

This file contains the logic and display of the begin game state (when the game is about to start. Notable features:

- I chose to initialize the mole field and hammer here, so the player could get their bearings before the game started
    - The hammer moves with the mouse, but the user cannot click to swing it

- There is also a collection of different sinister mole quotes I made up, stored in a list that get typed out on screen as the level loads, as if the mole is taunting the player

PlayState.lua

This file contains the logic and display of the play state (the game). I chose to abstract every non-logistical aspect of the game out of the play state, so the following are the only notable aspects that remain:

- Keeping track of score
    - Added function to allow other classes to not only change the score, but to visually draw attention to that change
- Activating moles at intervals with logic to enhance playability
- Keeping track of time

GameOverState.lua

This file contains the logic and display of the game over state. Notable features:

- Display the score from the last game
    - Indicate whether it is a new high score, and save to disk

Field.lua

This file contains the logic and display of the mole field. The field is made up of holes from which moles can emerge/be activated. It can also be skinned to any of four various themes:

- Desert
- Antarctica
- Hell
- Nuclear Wasteland

It contains a table of moles, corresponding to the various holes in the field, and can activate any amount requested by the play state, factoring in relevant constraints. It also handles the rendering of various overlay effects resulting from player interaction with the field (explosions, cracked glass, etc.)

Hammer.lua

This file contains the logic and display of the hammer. The hammer is used by the player to hit the moles. Notable features:

- The hammer is transparent until the user clicks to hit a mole to prevent it from blocking the player's view
- The hammer is skinned to the theme of the field to add variety
- The hammer has a natural movement effect that adjusts its rotation depending on speed and velocity as the player moves the mouse.

Mole.lua

This file contains the logic and display of moles. Moles persist throughout the life of the game, and can be activated and deactivated by being hit or retreating on their own. Notable features:

- A mole can be a ‘banner wielding mole’, and, if so, must be hit in ascending order relative to its fellow banner wielding moles
- A mole can also have an item that affects the player in some way, see the item section for this
- When a mole dies or retreats, it offloads the logic of that action to the respective Banner and Item classes where applicable

Banner.lua

This file contains the logic and display of banners, if a given mole is indeed a banner wielding mole. How it works:

- With probability, the play state requests 4 banner wielding moles from the field.
- If there are enough inactive moles, 4 moles are activated, designated as having a banner, and this class displays the banner corresponding to its mole’s banner number.
- To ensure that they are hit in order, this class has a table containing valid sums of remaining banner numbers, called BANNER_RULES.
    - Each value in BANNER RULES corresponds to the correct sum of remaining banner numbers at each index.
        - E.g, If a mole with a ‘1’ banner was hit, index 1 of this table indicates that the sum of the remaining banner numbers should be 9 (2 + 3 + 4).
- Banner moles have a longer delay before they retreat, allowing the user more time to hit them, as well as knock out bombs before they explode or tomatoes before they’re thrown.

Item.lua

This file contains the logic and display of items, if a given mole is indeed in possession of an item. The various item types are as follows:

1. Bomb - If this goes off, the game is immediately over
2. Tomato - Mole can throw this at the player's screen
3. Money Bag - Player is rewarded for hitting a mole with this
4. White Flag - Player is punished for hitting a mole with this

Crosshair.lua

This file contains the logic and display of the crosshair. The mouse is hidden throughout the game, and the crosshair is shown when input is allowed. Notable features:

- When changing between input allowed/disallowed, the crosshair is animated.

Note on effects: The bomb and the tomato both have visual effects when they are utilized by the mole. The bomb explodes and the tomato is thrown at the screen, cracking it temporarily. While I debated abstracting effects like these into an Effect.lua class, it felt forced considering how different the behaviors were and seemed to be unnecessarily abstract, bloating the code.