MAVERICK'S RUN

An Navigation Arcade Game
Built with Python

2021 Charles V Fisher for ITIN 8000

Problem: Campus Navigation

Students, staff, and faculty often find difficulty navigating the campus of the University of Nebraska at Omaha. Further efforts could be done to make tours, mapping, and building identification faster, more accessible, and fun. “Maverick’s Run” approaches this in the form of a fast paced arcade game designed for play on the Criss Library Creative Production Lab Arcade Cabinet.

Key Features

- Application will present a title screen and timer-idle attention grabber prior to interaction.
- Players can start by pressing a button on the keyboard or arcade cabinet.
- UNO campus will be rendered as a grid-based, 2-D, top down map.
- Players will spawn at an initial random location: parking lot, bus stop, or dorm.
- Players will be given a random building and room to travel to from their spawn location.
- A radial navigation arrow will point from the player avatar towards their target location.
- Players will move horizontally and vertically along the grid using WASD, arrows, or joystick.
- Players will be timed with an on-screen stopwatch from spawn to arrival at target location.
- On successful arrival at target location players can attach a 3 letter initial tag to their travel time.
- Travel times will be posted to a local-instance leaderboard.
- Players will be prompted to play again, exit to title, or timeout to title if left idle.

Issues of Consideration

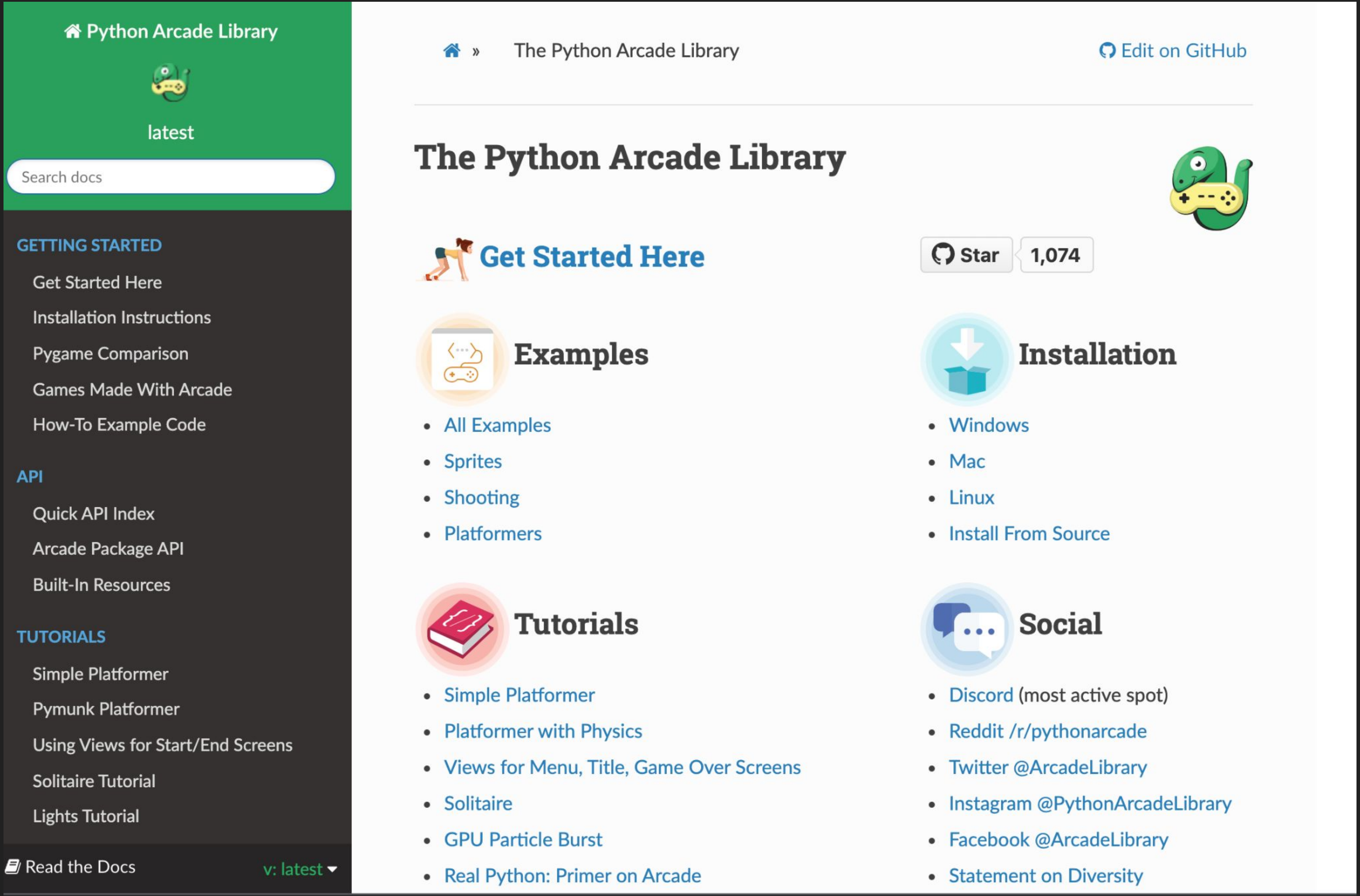
From a 2D game-map perspective, UNO is HUGE.

If a player character is 16pixels tall, a scaled version of just north campus is over 18,000 pixels wide. That’s going to take some processing...

Needed development ability to actively rework maps outside of code.

PyGame is great for small, single-screen games, but can become a bottleneck for large exploratory games.

Solution: Python Arcade!



Python Arcade Library

latest

Search docs

GETTING STARTED

- Get Started Here
- Installation Instructions
- Pygame Comparison
- Games Made With Arcade
- How-To Example Code

API

- Quick API Index
- Arcade Package API
- Built-In Resources

TUTORIALS

- Simple Platformer
- Pymunk Platformer
- Using Views for Start/End Screens
- Solitaire Tutorial
- Lights Tutorial

Read the Docs v: latest

» The Python Arcade Library

Edit on GitHub

The Python Arcade Library

Get Started Here

Star 1,074

Examples

- All Examples
- Sprites
- Shooting
- Platformers

Installation

- Windows
- Mac
- Linux
- Install From Source

Tutorials

- Simple Platformer
- Platformer with Physics
- Views for Menu, Title, Game Over Screens
- Solitaire
- GPU Particle Burst
- Real Python: Primer on Arcade

Social

- Discord (most active spot)
- Reddit /r/pythonarcade
- Twitter @ArcadeLibrary
- Instagram @PythonArcadeLibrary
- Facebook @ArcadeLibrary
- Statement on Diversity

Python Arcade Advantages

Phenomenal tutorials, examples, and API documentation.

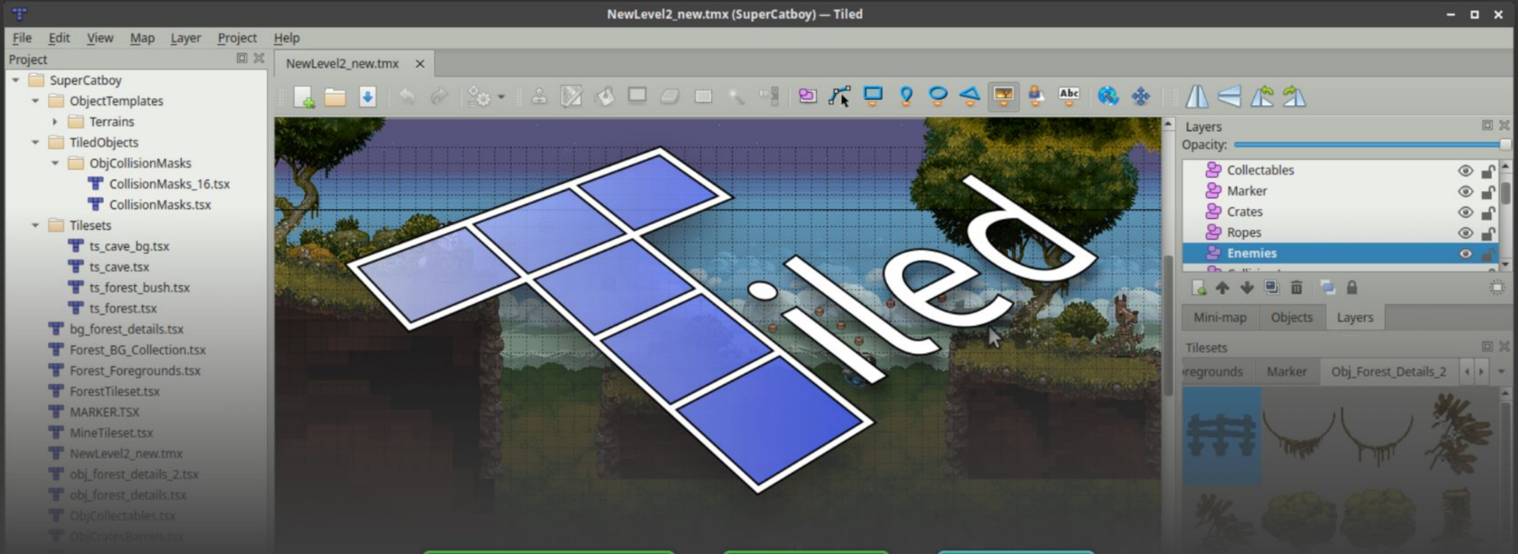
Scenes and SpriteLists: simple handling and packaging of all the things you'd want to draw on the screen with a single call

Built around pyglet: super fast handling of draw and update, plus easy sound handling

SUPER DUPER KEY FEATURE: native capacity to work with a program called Tiled for map builds.

Docs News Donate

Forum Discord Twitter GitHub



Download on itch.io Get Started Donate

Latest Updates

Tiled 1.7.2 Released 10 August 2021

Tiled 1.7.1 Released 11 July 2021

Working with Tiled

FREE program

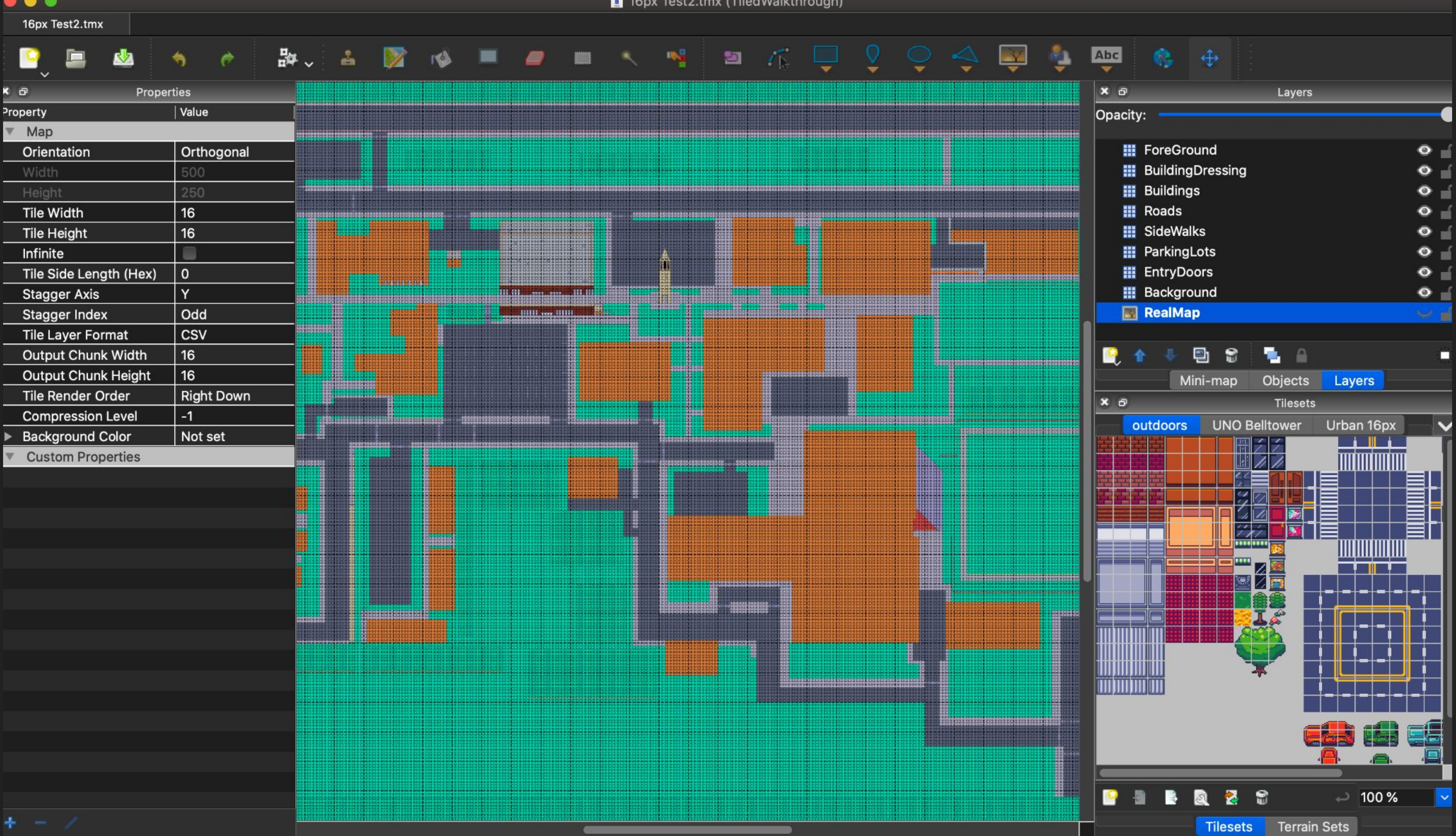
GUI based creation of maps with layers, collision control.

Layers: can set per-layer collision and draw-order. Create a background, buildings, tree-tops, etc and update them in one or two lines of code.

Exports to json for read-in by Arcade. Single Map Object!

Need to update map? Alter it in Tiled, export json to same location, and overwrite.

Easy back-and-forth iterative workflow



16px Test2.tmx (TiledWalkthrough)

16px Test2.tmx

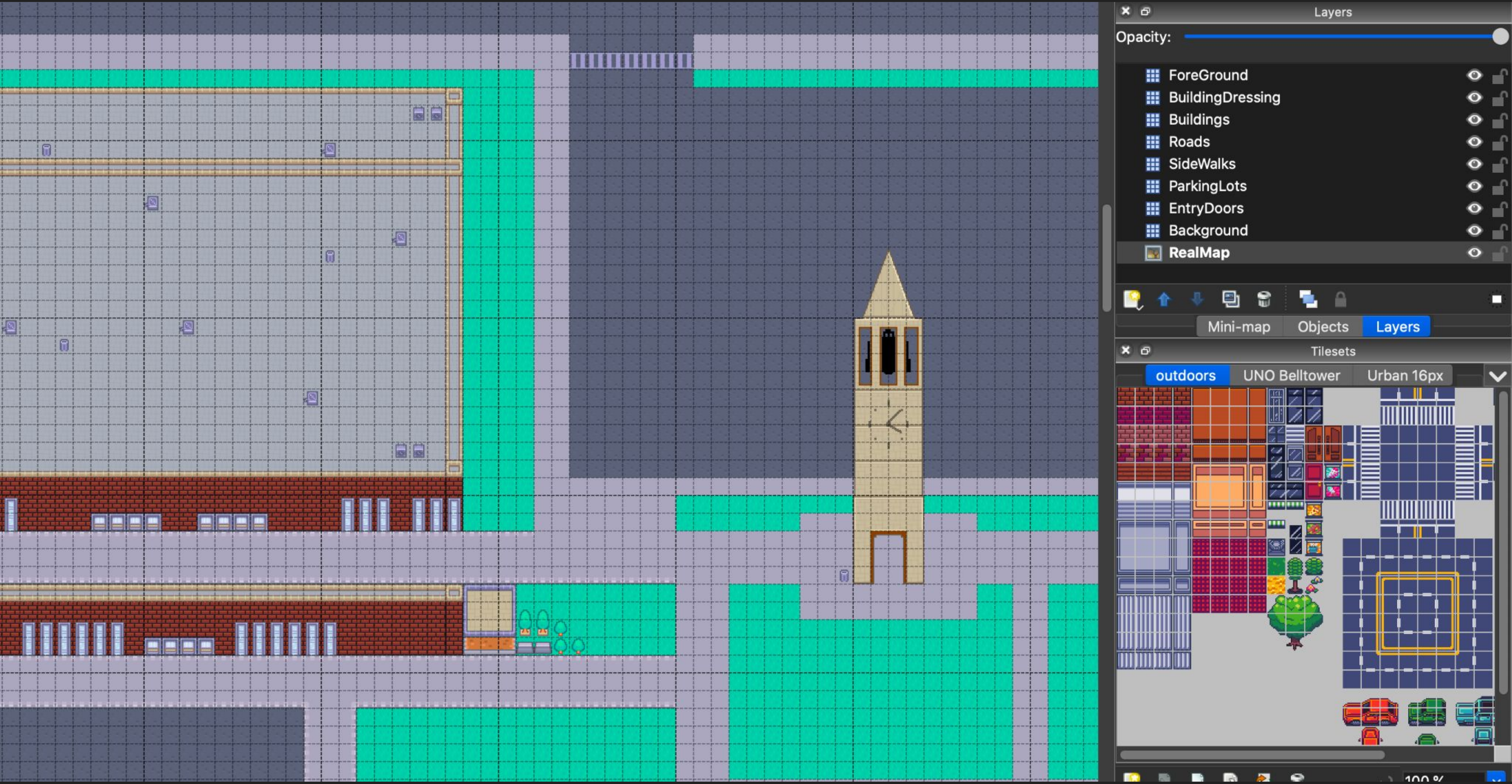
Properties

Property	Value
Map	
Orientation	Orthogonal
Width	500
Height	250
Tile Width	16
Tile Height	16
Infinite	☐
Tile Side Length (Hex)	0
Stagger Axis	Y
Stagger Index	Odd
Tile Layer Format	CSV
Output Chunk Width	16
Output Chunk Height	16
Tile Render Order	Right Down
Compression Level	-1
Background Color	Not set
Custom Properties	

Layers

Opacity:

- ForeGround
- BuildingDressing
- Buildings
- Roads
- SideWalks
- ParkingLots
- EntryDoors
- Background
- RealMap

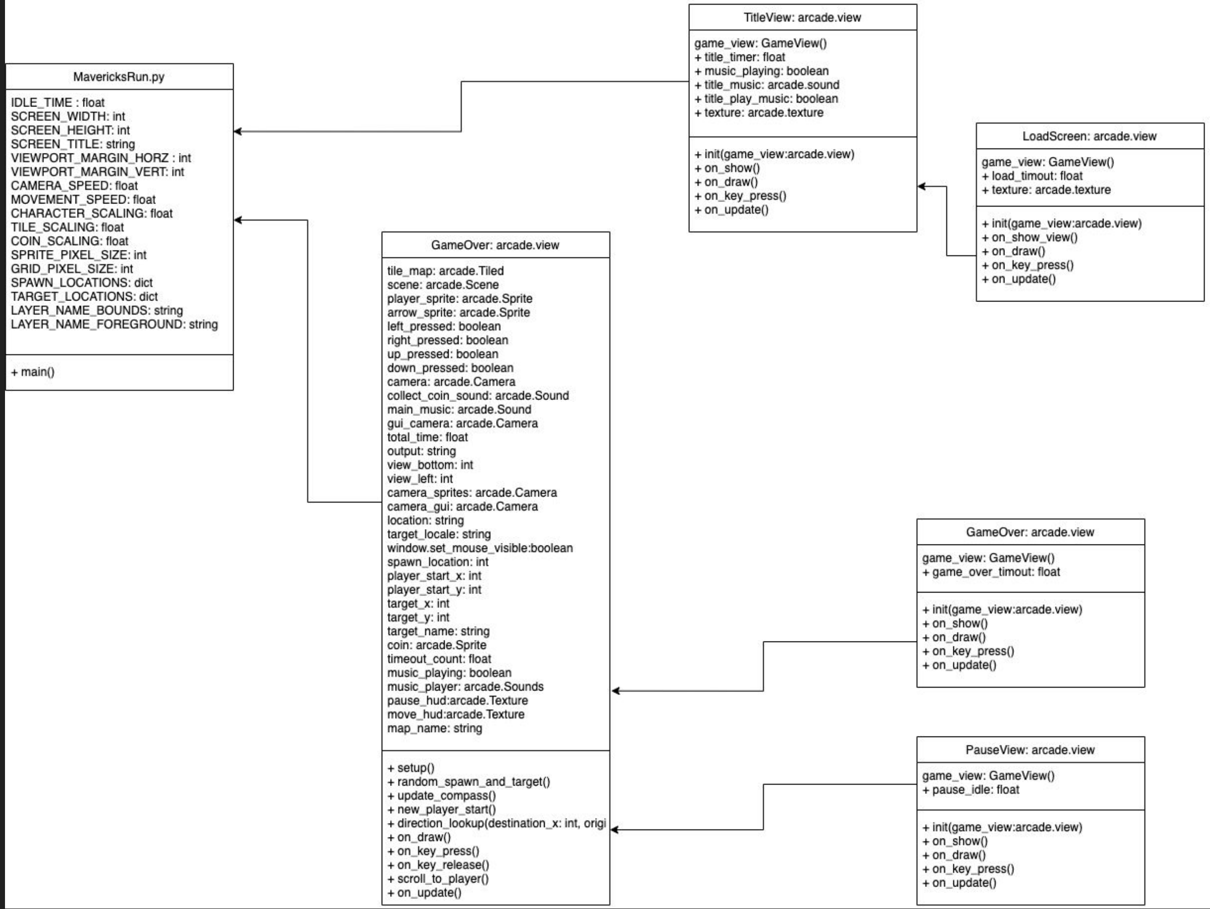

Layers
Opacity:
ForeGround
BuildingDressing
Buildings
Roads
SideWalks
ParkingLots
EntryDoors
Background
RealMap
Mini-map
Objects
Layers
Tilesets
outdoors
UNO Belltower
Urban 16px
100 %

Game Development in Python

PRD laid out some larger challenges:

- Multiple “scenes”
- Gigantic map of sprites to load in
- Radial compass
- Randomized start and target locations

Maverick's Run UML
Charles V Fisher
2021 ITIN 8000



MavericksRun.py

IDLE_TIME : float
SCREEN_WIDTH: int
SCREEN_HEIGHT: int
SCREEN_TITLE: string
VIEWPORT_MARGIN_HORZ : int
VIEWPORT_MARGIN_VERT: int
CAMERA_SPEED: float
MOVEMENT_SPEED: float
CHARACTER_SCALING: float
TILE_SCALING: float
COIN_SCALING: float
SPRITE_PIXEL_SIZE: int
GRID_PIXEL_SIZE: int
SPAWN_LOCATIONS: dict
TARGET_LOCATIONS: dict
LAYER_NAME_BOUNDS: string
LAYER_NAME_FOREGROUND: string

+ main()

TitleView: arcade.view

game_view: GameView()
+ title_timer: float
+ music_playing: boolean
+ title_music: arcade.sound
+ title_play_music: boolean
+ texture: arcade.texture

+ init(game_view:arcade.view)
+ on_show()
+ on_draw()
+ on_key_press()
+ on_update()

LoadScreen: arcade.view

game_view: GameView()
+ load_timout: float
+ texture: arcade.texture

+ init(game_view:arcade.view)
+ on_show_view()
+ on_draw()
+ on_key_press()
+ on_update()

GameOver: arcade.view

tile_map: arcade.Tiled
scene: arcade.Scene
player_sprite: arcade.Sprite
arrow_sprite: arcade.Sprite
left_pressed: boolean
right_pressed: boolean
up_pressed: boolean
down_pressed: boolean
camera: arcade.Camera
collect_coin_sound: arcade.Sound
main_music: arcade.Sound
gui_camera: arcade.Camera
total_time: float
output: string
view_bottom: int
view_left: int
camera_sprites: arcade.Camera
camera_gui: arcade.Camera
location: string
target_locale: string
window.set_mouse_visible:boolean
spawn_location: int
player_start_x: int
player_start_y: int
target_x: int
target_y: int
target_name: string
coin: arcade.Sprite
timeout_count: float
music_playing: boolean
music_player: arcade.Sounds
pause_hud:arcade.Texture
move_hud:arcade.Texture
map_name: string

+ setup()
+ random_spawn_and_target()
+ update_compass()
+ new_player_start()
+ direction_lookup(destination_x: int, origi
+ on_draw()
+ on_key_press()
+ on_key_release()
+ scroll_to_player()
+ on_update()

GameOver: arcade.view

game_view: GameView()
+ game_over_timout: float

+ init(game_view:arcade.view)
+ on_show()
+ on_draw()
+ on_key_press()
+ on_update()

PauseView: arcade.view

game_view: GameView()
+ pause_idle: float

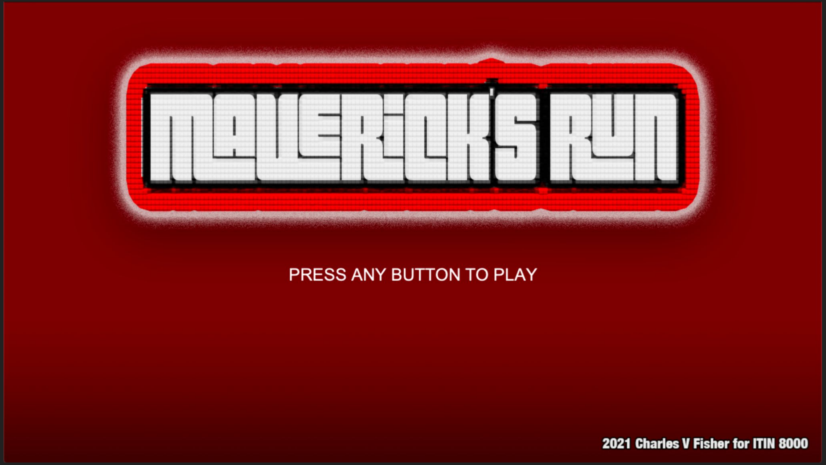
+ init(game_view:arcade.view)
+ on_show()
+ on_draw()
+ on_key_press()
+ on_update()

“Scenes” via arcade.Views

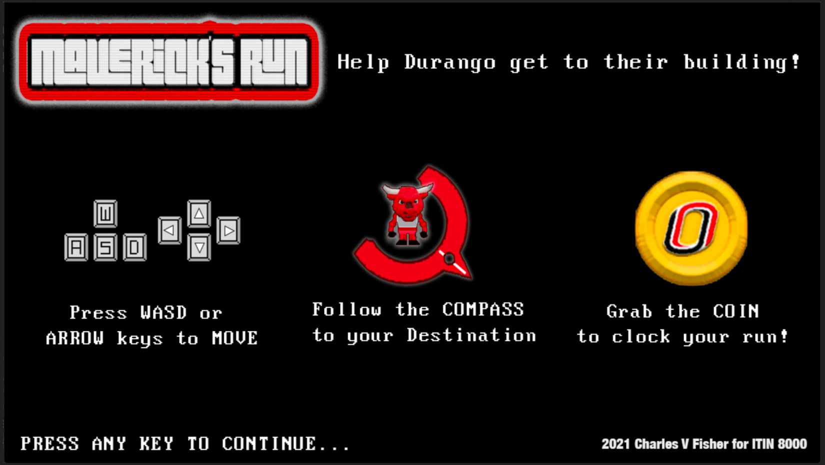
Python arcade has handy dandy “Views” class that does things quickly: bundles up the things you want to draw on screen into a short chunk of code, then let’s you call instantiated views “to the front” with a single line call.

As per UML above, main() creates a TitleScreen and GameView to swap between. TitleScreen has a child LoadScreen, but both are aware of the GameView in order to easily swap to it.

GameView has two children: PauseView and GameOver. This enables easy pause and restart of the main game loop, or simple swapback to TitleScreen.


PRESS ANY BUTTON TO PLAY

2021 Charles V Fisher for ITIN 8000


Help Durango get to their building!

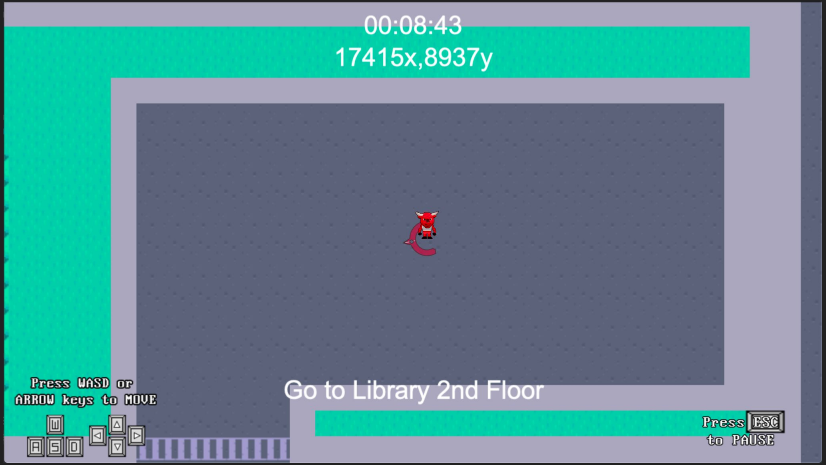
Press WASD or ARROW keys to MOVE

Follow the COMPASS to your Destination

Grab the COIN to clock your run!

PRESS ANY KEY TO CONTINUE...

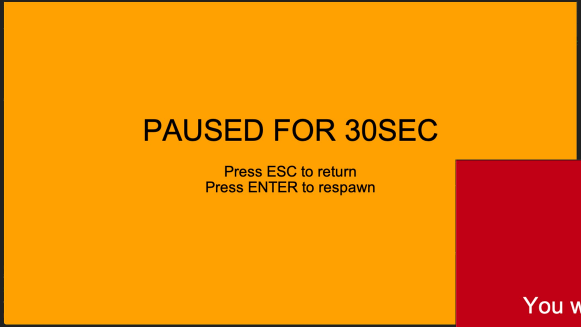
2021 Charles V Fisher for ITIN 8000


00:08:43
17415x,8937y

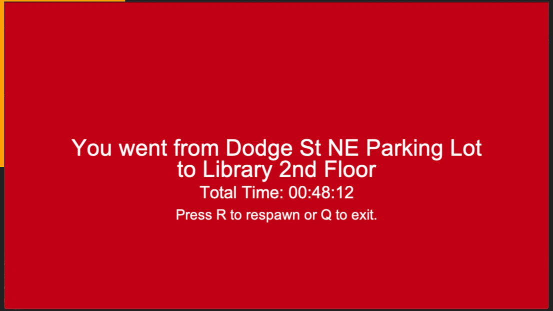
Go to Library 2nd Floor

Press WASD or ARROW keys to MOVE

Press ESC to PAUSE


PAUSED FOR 30SEC

Press ESC to return
Press ENTER to respawn


You went from Dodge St NE Parking Lot to Library 2nd Floor

Total Time: 00:48:12

Press R to respawn or Q to exit.

Gigantic Map? Hide the setup and load!

Early builds didn’t load the map of campus until the same time as generating the player start point. DUMB. Had to load the json, recreate the sprite list and redraw the whole thing EVERY SINGLE RESPAWN.

arcade.Views solved this!

GameView’s setup() call can happen as part of main(), meaning everything is loaded and ready before the TitleView screen shows up.

Problems with collision or map layout? Edit in Tiled and reload. 30sec-1min fix.

Radial Compass. Small math but proud work.

Math time! Take the XY of where you are, and the XY of where you want to get, and use python’s math library to aTan() their deltas. This gives you back a radian.

Convert that radian return to an angle 0-360

arcade.sprite.angle() lets you rotate a currently drawn sprite.

Since you’re always updating the player’s XY, you can use a separate “compass” sprite to point the direction to your destination by rotating it!

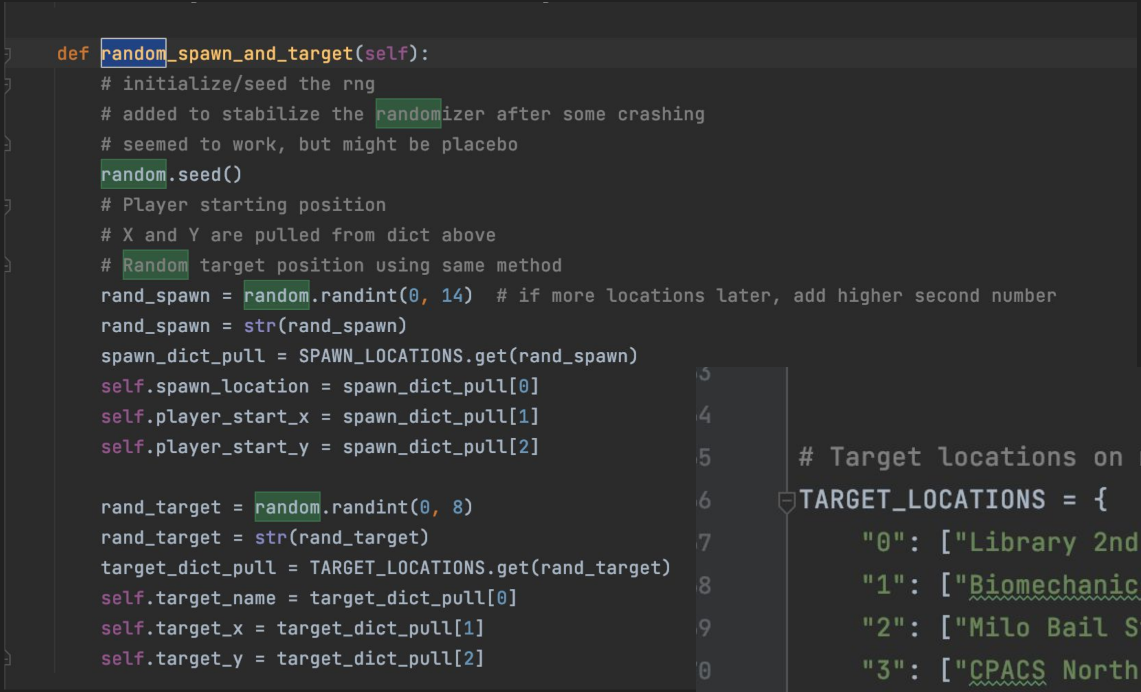
Ever so random: Spawn and Target

Simple but expandable: Create a dict in the CONSTANTS.

Predefine a location as a string “Name” and their X Y as a list stored in the dict.

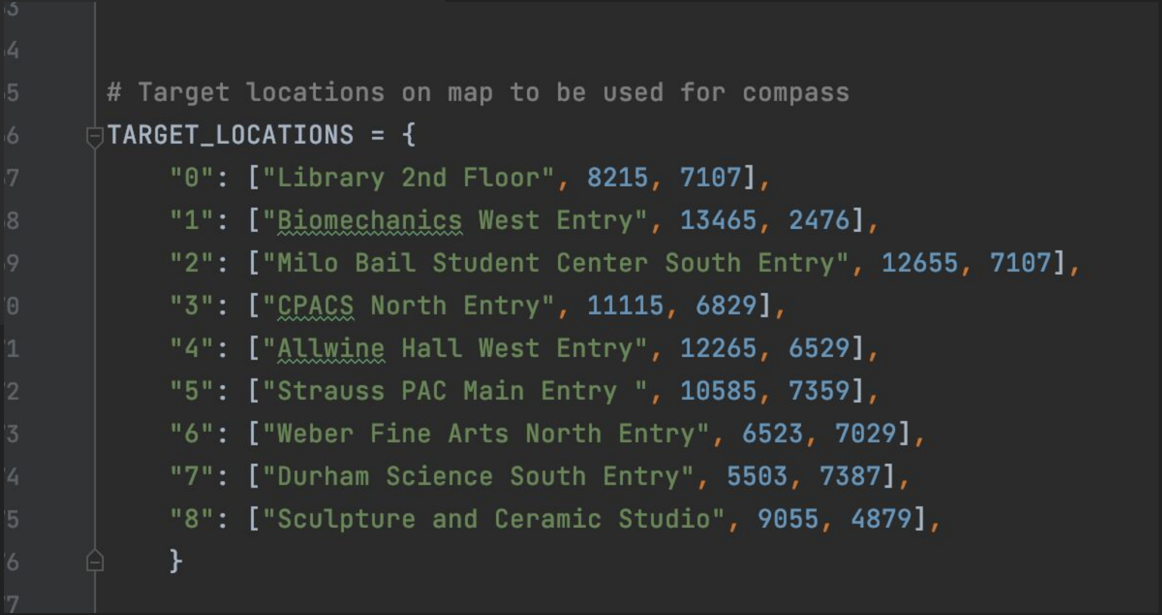
.get() each piece as part of GameView variables.

Next Level Mode: Feed all this in as a .csv, and let someone else make new locations that can easily be added in the new_player_start() call!



```python
def random_spawn_and_target(self):
    # initialize/seed the rng
    # added to stabilize the randomizer after some crashing
    # seemed to work, but might be placebo
    random.seed()
    # Player starting position
    # X and Y are pulled from dict above
    # Random target position using same method
    rand_spawn = random.randint(0, 14)  # if more locations later, add higher second number
    rand_spawn = str(rand_spawn)
    spawn_dict_pull = SPAWN_LOCATIONS.get(rand_spawn)
    self.spawn_location = spawn_dict_pull[0]
    self.player_start_x = spawn_dict_pull[1]
    self.player_start_y = spawn_dict_pull[2]

    rand_target = random.randint(0, 8)
    rand_target = str(rand_target)
    target_dict_pull = TARGET_LOCATIONS.get(rand_target)
    self.target_name = target_dict_pull[0]
    self.target_x = target_dict_pull[1]
    self.target_y = target_dict_pull[2]
```



```python
# Target locations on map to be used for compass
TARGET_LOCATIONS = {
    "0": ["Library 2nd Floor", 8215, 7107],
    "1": ["Biomechanics West Entry", 13465, 2476],
    "2": ["Milo Bail Student Center South Entry", 12655, 7107],
    "3": ["CPACS North Entry", 11115, 6829],
    "4": ["Allwine Hall West Entry", 12265, 6529],
    "5": ["Strauss PAC Main Entry ", 10585, 7359],
    "6": ["Weber Fine Arts North Entry", 6523, 7029],
    "7": ["Durham Science South Entry", 5503, 7387],
    "8": ["Sculpture and Ceramic Studio", 9055, 4879],
    }
```

Drawbacks and Problems

While arcade is built around pyglet...it introduces some issues, namely around compiling an executable.

pyglet is small and has no dependencies. Easy to use pyinstaller and create a standalone executable in about 2 minutes.

Arcade has a number of nested resources and dependencies that just sort of...hang up when trying a simple pyinstaller compilation, or outright junk-builds with py2app.

Warrants more investigation, but none of the examples or available games built in arcade actually have a compiled version...

More Tidbits

The current structure makes it easy to swap out a player sprite and map, making this something that could work well for travelling around other areas, or even historical maps of campus.

External collaborators would only need to use Tiled to create a map from current tilesets, and update the locations dictionaries by walking through their map and naming places.

More Tidbits

Stable, playable, and simple...but needs more of an interaction hook in form of rewards and obstacles: hidden coins, functional leaderboard, cars to avoid...a menacing pursuer?

In building the game the importance of the map's visuals became increasingly necessary. The need for consistent grammar of collision areas and real-world landmarks was made clear the instant the Bell Tower was implemented.

Obvious next step is "juice": animation of player sprite, bump sounds, hidden areas, animated map components, etc.