Maps and Tiles

- Admin
- Questions?
- Schedule?
- Projects/Assignments?

Creating backgrounds

- We've looked at creating scrolling backgrounds
  - using large images
- other backgrounds
  - particularly for building worlds
  - want to build large world feel
    - minimal art resources

Inspiration from history

- The Greeks and Romans did it

Mosaic concept

- Use lots of tiles
  - more or less identical sized
  - limited number of different colors
  - create lots of different images

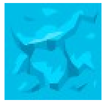
Use same concept in games

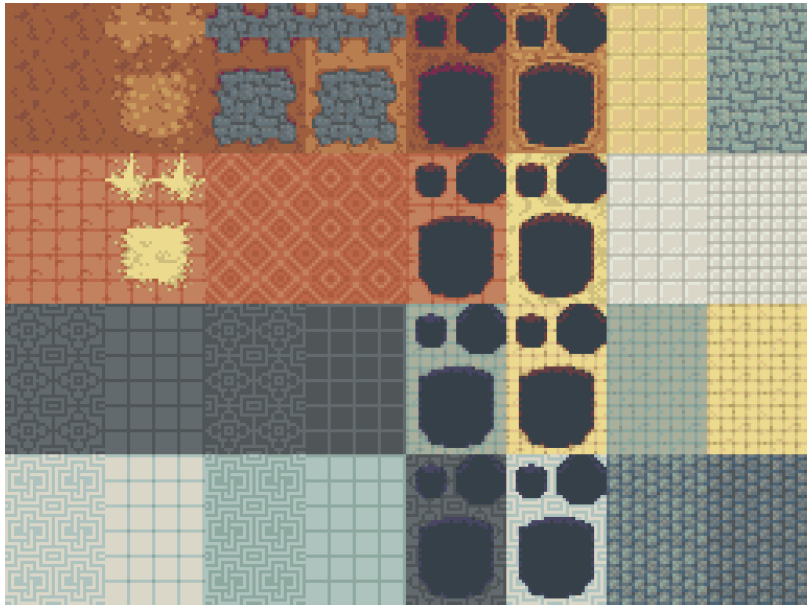
- Using mosaic concept in games
  - tiles are now images
    - identical sizes
    - relatively small number of images
  - use a combination of images to build a “world”
    - use different combination of same images to create different world

Simple approach

- Simple tiled maps can be setup
  - sort of 'paint by number' approach
  - create 'map' in text file
    - read in text file to load map into memory
    - each character in text file refers to a particular image
      - place image into window in appropriate grid location
  - will look at steps

Get images for world building

- Some old simple images:
- 
-
- Or the one file approach:
  - On web second is best
  - Locally, which ever
  - Why?



Text File Approach

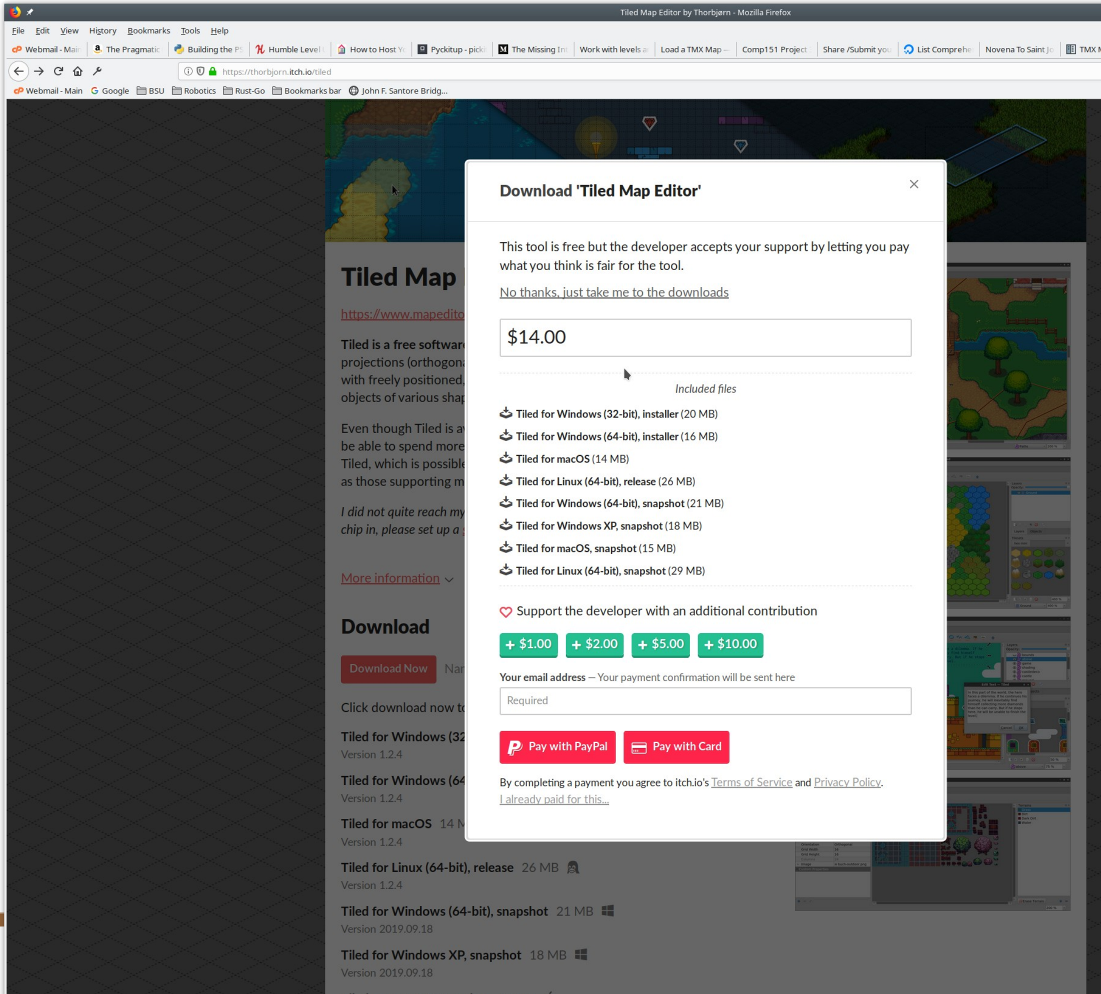
- Choose a character to represent an image/tile in 1-1 relationship
  - even in ascii 100+ characters
  - build the file
    - sample:
      - BBBBBB
      - BDDDDB
      - DDDDDB
      - BGGGGB
      - BGGGGB
      - BBBBBB
    - B= Brick; D= dirt; G = grass

Library Tiled map support

- In the old days we would read in this text file, then parse the characters out in a big embedded for loop.
- There are libraries now that build in support for tiled maps so we don’t have to do it ourselves now.
- Supports map files built using mapeditor
  - https://www.mapeditor.org/
- Build it yourself and skip the please pay screen: https://github.com/bjorn/tiled.git
- Or:

Tiled/MapEditor

- Get the prebuilt version
- My recommendation:
  - Download now as poor student
  - If you still use it after graduation, then kick in a few bucks.



Tiled Map Editor by Thorbjørn - Mozilla Firefox

https://thorbjorn.itch.io/tiled

Download 'Tiled Map Editor'

This tool is free but the developer accepts your support by letting you pay what you think is fair for the tool.

No thanks, just take me to the downloads

$14.00

Included files

- Tiled for Windows (32-bit), installer (20 MB)
- Tiled for Windows (64-bit), installer (16 MB)
- Tiled for macOS (14 MB)
- Tiled for Linux (64-bit), release (26 MB)
- Tiled for Windows (64-bit), snapshot (21 MB)
- Tiled for Windows XP, snapshot (18 MB)
- Tiled for macOS, snapshot (15 MB)
- Tiled for Linux (64-bit), snapshot (29 MB)

Support the developer with an additional contribution

+ $1.00 + $2.00 + $5.00 + $10.00

Your email address — Your payment confirmation will be sent here

Required

Pay with PayPal Pay with Card

By completing a payment you agree to itch.io's Terms of Service and Privacy Policy.
I already paid for this...

Download

Download Now

Tiled for Linux (64-bit), release 26 MB
Version 1.2.4

Tiled for Windows (64-bit), snapshot 21 MB
Version 2019.09.18

Tiled for Windows XP, snapshot 18 MB
Version 2019.09.18

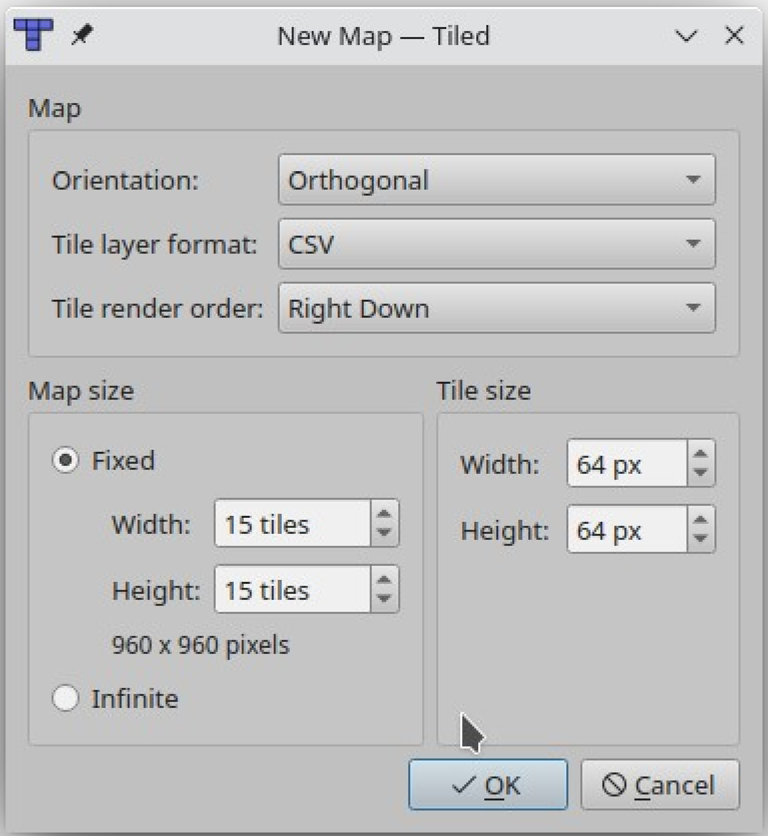
Tiled/MapEditor

- Grab some images from somewhere
  - Make sure they are all the same size
  - Typical sizes are 72x72 and various powers of 2
    - 32x32
    - 64x64
    - 128x128
    - 256x256

- You can find a small set of very simple tiles in the demo on github.
-

Create A New Map

- When running tiled, first create new map
  - Orientation: our library only promises Orthogonal (top down) will work
  - Choose a fixed map size of your choice
  - Adjust tile size to be the size of your images.


New Map — Tiled

Map

Orientation: Orthogonal

Tile layer format: CSV

Tile render order: Right Down

Map size

[✓ Fixed]

Width: 15 tiles

Height: 15 tiles

960 x 960 pixels

Infinite

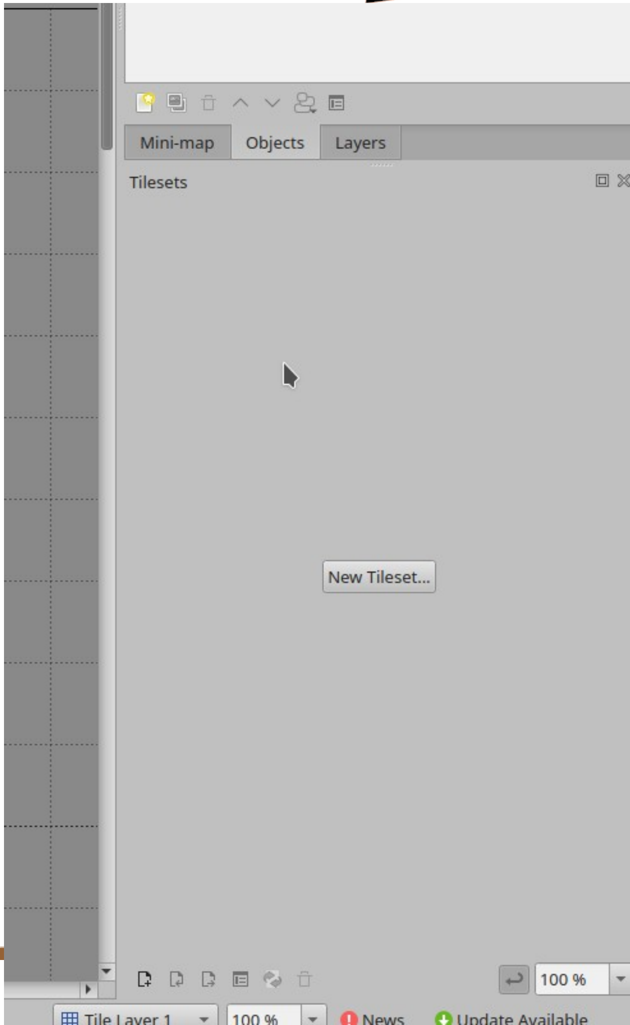
Tile size

Width: 64 px

Height: 64 px

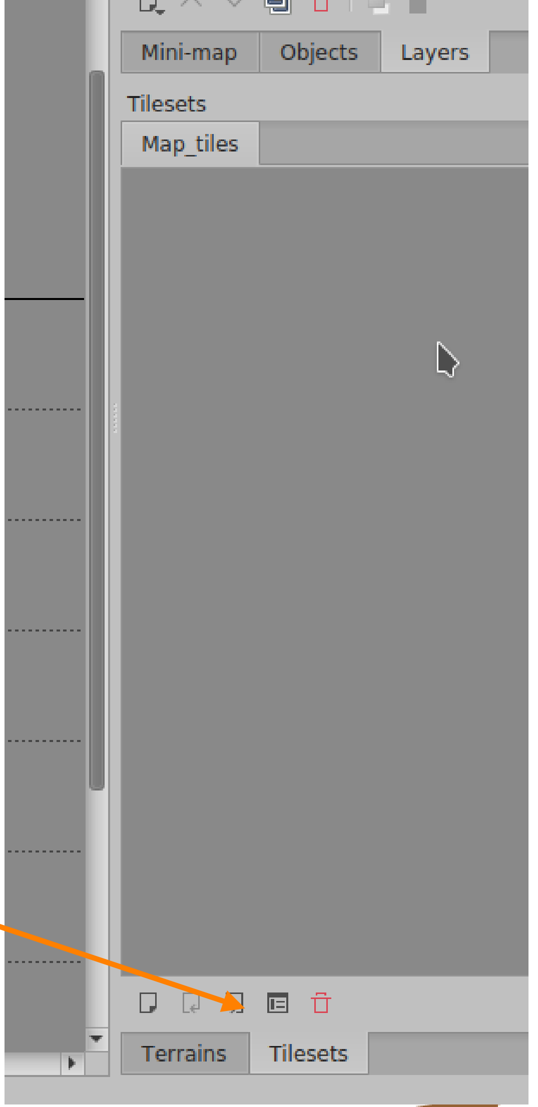
TileSets

- Now that we have a map, we need some tiles
  - Tiled supports tileset images
    - Single images with many tiles embedded in them
  - And a collection of individual images.
  - Choose <file><new tileset>
  - Or hit the <New Tileset> button in the section to the bottom right of the screen---------------------->
  - We'll choose
    - "Collection of images" and
    - "Embed in map"



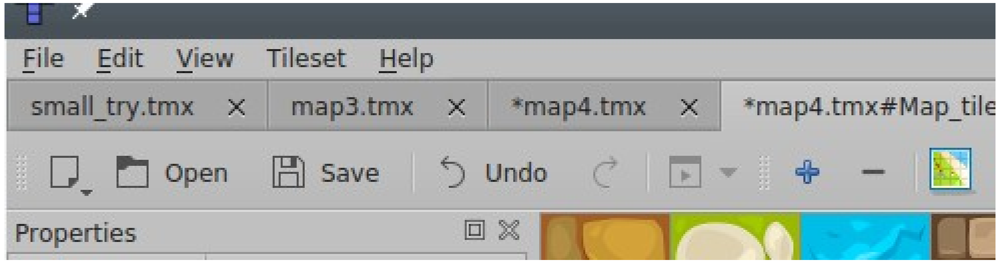
Tileset with no tiles

- Now we have a tileset with no tiles.
  - We need to add them.
  - Choose edit tileset
  - Which is hard to find



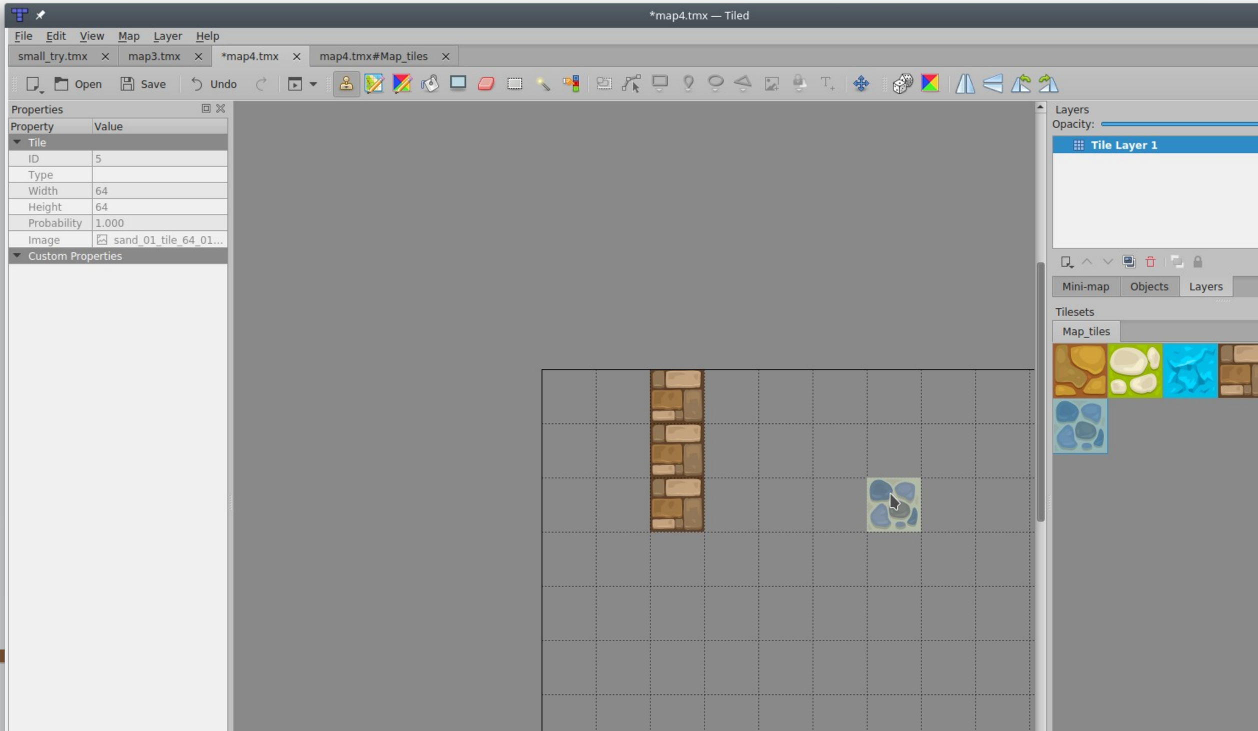
Add new tile to tileset

- Now we need to add tiles to tileset.
  - Press the "+" button
  - From the resulting file dialog select your images
  - I recommend having them in your assets folder already
    - A subdirectory of your project where you will save this map as well
  - Then select your map.
    - It will be a tab in tiled.



Build Basic map

- Now to build a basic map
  - Use the tiles to paint the map with and save the map



Properties

Property	Value
Tile	
ID	5
Type	
Width	64
Height	64
Probability	1.000
Image	sand_01_tile_64_01...
Custom Properties	

Very Simple first program

- For the last time, we will dump all of the files into the main project folder
- Get the zip file from the resources page to follow along.
- The starting code is here:→
  - More coming next

```go
import (
	"fmt"
	"github.com/hajimehoshi/ebiten/v2"
	"github.com/hajimehoshi/ebiten/v2/ebitenutil"
	"github.com/lafriks/go-tiled"
	"os"
)
```

```go
const mapPath = "demoMap.tmx" // Path to your Tiled Map.

type mapGame struct {
	Level    *tiled.Map
	tileHash map[uint32]*ebiten.Image
}

func (m mapGame) Update() error {
	return nil
}

func (m mapGame) Layout(outsideWidth, outsideHeight int)
(screenWidth, screenHeight int) {
	//TODO implement me
	return outsideWidth, outsideHeight
}
```

Main function

```
func main() {
	// Parse .tmx file.
	gameMap, err := tiled.LoadFile(mapPath)
	windowWidth := gameMap.Width * gameMap.TileWidth
	windowHeight := gameMap.Height * gameMap.TileHeight
	ebiten.SetWindowSize(windowWidth, windowHeight)
	if err != nil {
		fmt.Printf("error parsing map: %s", err.Error())
		os.Exit(2)
	}
	ebitenImageMap := makeEbiteImagesFromMap(*gameMap)
	oneLevelGame := mapGame{
		Level:    gameMap,
		tileHash: ebitenImageMap,
	}
	fmt.Println("tilesets:", gameMap.Tilesets[0].Tiles)
	//fmt.Println("layers:", gameMap.Layers[0].Tiles)
	fmt.Print("type:", fmt.Sprintf("%T", gameMap.Layers[0].Tiles[0]))
	err = ebiten.RunGame(&oneLevelGame)
	if err != nil {
		fmt.Println("Couldn't run game:", err)
	}
}
```

```
func makeEbiteImagesFromMap(tiledMap tiled.Map)
map[uint32]*ebiten.Image {
	idToImage := make(map[uint32]*ebiten.Image)
	for _, tile := range tiledMap.Tilesets[0].Tiles {
		ebitenImageTile, _, err :=
ebitenutil.NewImageFromFile(tile.Image.Source)
		if err != nil {
			fmt.Println("Error loading tile image:",
tile.Image.Source, err)
		}
		idToImage[tile.ID] = ebitenImageTile
	}
	return idToImage
}
```

And finally Draw

```
func (game mapGame) Draw(screen *ebiten.Image) {
	drawOptions := ebiten.DrawImageOptions{}
	for tileY := 0; tileY < game.Level.Height; tileY += 1 {
		for tileX := 0; tileX < game.Level.Width; tileX += 1 {
			drawOptions.GeoM.Reset()
			TileXpos := float64(game.Level.TileWidth * tileX)
			TileYpos := float64(game.Level.TileHeight * tileY)
			drawOptions.GeoM.Translate(TileXpos, TileYpos)
			tileToDraw :=
game.Level.Layers[0].Tiles[tileY*game.Level.Width+tileX]
			ebitenTileToDraw := game.tileHash[tileToDraw.ID]
			screen.DrawImage(ebitenTileToDraw,
&drawOptions)
		}
	}
}
```

- Notice that we are drawing each tile one by one
  - If an engine hides this, it is still being done
  - Is there any way around this?
  - If we haven't done this before
    - Let's put this into goland and see it work.

Efficiency: games need it.

- For efficiency:
  - create a new image
  - Draw the map onto the image once
  - Just draw that one image till the end of the game
  - Update the game to include a third member the image for the map background.

```
type mapGame struct {
    Level        *tiled.Map
    tileHash     map[uint32]*ebiten.Image
    drawableLevel *ebiten.Image
}
```

- Then draw the map to this image

```
func buildDrawableLevel(game *mapGame) {
        screen := game.drawableLevel
        drawOptions := ebiten.DrawImageOptions{}
        for tileY := 0; tileY < game.Level.Height; tileY += 1 {
                for tileX := 0; tileX < game.Level.Width; tileX += 1 {
                        drawOptions.GeoM.Reset()
                        TileXpos := float64(game.Level.TileWidth * tileX)
                        TileYpos := float64(game.Level.TileHeight * tileY)
                        drawOptions.GeoM.Translate(TileXpos, TileYpos)
                        tileToDraw :=
game.Level.Layers[0].Tiles[tileY*game.Level.Width+tileX]
                        ebitenTileToDraw := game.tileHash[tileToDraw.ID]
                        screen.DrawImage(ebitenTileToDraw,
&drawOptions)
                }
        }
}
```

- And now draw is just two lines

Complete

- See for the complete rewrite
- https://github.com/jsantore/MapDemoOneImage

Go:embed

- Amazing added feature in go.
  - You can embed files directly into the go program (the final executable) so all you have to give someone is a single executable file
  - Text files, images etc

Go:embed II

- Usage
- You need the go:embed directive in a comment immediately over a global (or at least package wide) variable, which will hold the embedded asset
- Eg:

-
  ```
  //go:embed assets/*
  var EmbeddedAssets embed.FS
  ```
- This will take everything in the assets subfolder of the project and treat it as a file system

Go:Embed III

- Example function to open an image from embedded file system.
- Haven't tried to be 'clever'
  - Still have one function per file type

- Example load function

```
func loadPNGImageFromEmbedded(name string) *ebiten.Image {
        embeddedFile, err := EmbeddedAssets.Open("assets/" +
name)
        if err != nil {
                log.Fatal("failed to load embedded image ",
embeddedFile, err)
        }
        rawImage, err := png.Decode(embeddedFile)
        if err != nil {
                log.Fatal("failed to load embedded image ", name, err)
        }
        gameImage := ebiten.NewImageFromImage(rawImage)
        return gameImage
}
```

So now we will use folders

- Ok, so now we will never put our assets into the main folder again?
- Why?

Now lets build the map embedded

- Make a new goland project
  - Make a folder called assets
  - Unzip the zip file we got for our last demo into the assets folder.
  - Make a new go file
    - Now we are ready to begin.

- The start of the file:

```go
package main

import (
	"embed"
	"fmt"
	"github.com/hajimehoshi/ebiten/v2"
	"github.com/hajimehoshi/ebiten/v2/ebitenutil"
	"github.com/lafriks/go-tiled"
	"log"
	"path"
)

//go:embed assets/*
var EmbeddedAssets embed.FS

type mapGame struct {
	Level    *tiled.Map
	tileHash map[uint32]*ebiten.Image
}
```

The ebiten.Game interface

- The three methods required by the ebiten.Game interface are the same as for the naive implementation
  - You can copy them from the previous version.

```
func (m mapGame) Update() error {
        return nil
}

func (game mapGame) Draw(screen *ebiten.Image) {
        drawOptions := ebiten.DrawImageOptions{}
        for tileY := 0; tileY < game.Level.Height; tileY += 1 {
                for tileX := 0; tileX < game.Level.Width; tileX += 1 {
                        drawOptions.GeoM.Reset()
                        TileXpos := float64(game.Level.TileWidth * tileX)
                        TileYpos := float64(game.Level.TileHeight * tileY)
                        drawOptions.GeoM.Translate(TileXpos, TileYpos)
                        tileToDraw :=
game.Level.Layers[0].Tiles[tileY*game.Level.Width+tileX]
                        ebitenTileToDraw := game.tileHash[tileToDraw.ID]
                        screen.DrawImage(ebitenTileToDraw, &drawOptions)
                }}}

func (m mapGame) Layout(outsideWidth, outsideHeight int) (screenWidth,
screenHeight int) {
        return outsideWidth, outsideHeight
}
```

Main

- Main function primarily differs in which functions are called to load map/images
-

```
func main() {
	gameMap := loadMapFromEmbedded(path.Join("assets",
"demoMap.tmx"))

ebiten.SetWindowSize(gameMap.TileWidth*gameMap.Width,
gameMap.TileHeight*gameMap.Height)
	ebiten.SetWindowTitle("Maps Embedded")
	ebitenImageMap :=
makeEbiteImagesFromMap(*gameMap)
	oneLevelGame := mapGame{
		Level:    gameMap,
		tileHash: ebitenImageMap,
	}
	err := ebiten.RunGame(&oneLevelGame)
	if err != nil {
		fmt.Println("Couldn't run game:", err)
	}
}
```

Loading the map

- Big change is how map is loaded and images
- Let's look at this then try it

-
  ```
  func loadMapFromEmbedded(name string) *tiled.Map {
  	embeddedMap, err := tiled.LoadFile(name,
  tiled.WithFileSystem(EmbeddedAssets))
  	if err != nil {
  		fmt.Println("Error loading embedded map:",
  err)
  	}
  	return embeddedMap
  }
  ```
- See the whole thing here:
- https://github.com/shinjitsu/TiledWithEmbed

```
func makeEbiteImagesFromMap(tiledMap tiled.Map)
map[uint32]*ebiten.Image {
	idToImage := make(map[uint32]*ebiten.Image)
	for _, tile := range tiledMap.Tilesets[0].Tiles {
	embeddedFile, err := EmbeddedAssets.Open(path.Join("assets",
tile.Image.Source))
	if err != nil {
	log.Fatal("failed to load embedded image ", embeddedFile, err)
	}
	ebitenImageTile, _, err :=
ebitenutil.NewImageFromReader(embeddedFile)
	if err != nil {
	fmt.Println("Error loading tile image:", tile.Image.Source, err)
	}
	idToImage[tile.ID] = ebitenImageTile
	}
	return idToImage
}
```

Questions?