Farming out Map Collisions

Admin

- Quizzes
- Schedule
- Questions?

Tiles and collisions

- I was asked a while ago how we could automate/outsource our collision checks when using a tiled map
- I’ve used a library for this last fall on sabbatical for a bigger project
  - So I figured – sure we can do that
- Unfortunately it went public archive in November
  - But we’ll use it anyway.
- https://github.com/setanarut/tilecollider

TileCollider

- This library is intended to work with ebitengine
  - But was not necessarily designed to work with the go-tiled library
    - So we will have to make some adjustements

TileCollider and go-tiled

- Adjustment 1:
  - go-tiled unrolls the map into a 1d slice of map tiles
  - TileCollider wants a slice of slices (a 2d array if you will)
  - So we need to build the 2d slice from the unrolled 1d slice (reroll it)

TileCollider and go-tiled

- Adjustment 2:
  - go-tiled auto adjusts the map tileIDs so that the tile listed as ‘1’ in the map file is represented as ‘0’ in the 1D slice representing the map
    - And otherwise has an off by one issue 2 becomes 1 and so on
  - TileCollider wants all traversable tiles to be ‘0’ while non-traversable tiles can be any positive integer
  - So we need to adjust the numbers as we parse the file and build the TileCollider part of the map

Demo Start

- To demo a tile collider, I need a tiled map to start with
  - I began with the demo of a tiled map that gets painted on to a single image
    - https://github.com/shinjitsu/MapDemoOneImage
  - Which we have already worked through
  - So I'll start with that and only look at what is new

The Structs

- Game struct gets two new fields
  - Highlighted
- New player struct
  - Since we have to have something to collide

```
type mapGame struct {
        Level       *tiled.Map
        tileHash    map[uint32]*ebiten.Image
        drawableLevel *ebiten.Image
        collider    *tilecollider.Collider[int]
        demoPlayer  player
}

type player struct {
        x, y float64
        pict *ebiten.Image
}
```

```
func makeCollideMap(gameMap *tiled.Map) [][]int {
	//here my map has one layer, in a more realistic example I might have the
	//0 layer be the ground and then the next layer have only the obstacles
	//the tilemap is unrolled, with the 2d array unrolled into a 1d array
	//we have to convert it to a 2d array for the tilecollider to work
	var mapAsIntSlice [][]int = make([][]int, gameMap.Height)
	for tileY := 0; tileY < gameMap.Height; tileY += 1 {
		//get each row of tiles
		mapAsIntSlice[tileY] = make([]int, gameMap.Width)
		for tileX := 0; tileX < gameMap.Width; tileX += 1 {
			mapTileID := int(gameMap.Layers[0].Tiles[tileY*gameMap.Width+tileX].ID)
			//the tile collider wants 0 for all open tiles and non-zero for all obstacles
			//I want the brown tiles to be the obstacles, in the map they are 1 and 4, because of the
			//way the tiled library adjusts by one, they will be 0 and 3 in the array
			//those will be zero, the rest will be 1
			if mapTileID == 0 || mapTileID == 3 {
				mapTileID = 1
			} else {
				mapTileID = 0
			}
			mapAsIntSlice[tileY][tileX] = mapTileID
		}
	}
	return mapAsIntSlice
}
```

```
func main() {
	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)
	playerPict, _, err := ebitenutil.NewImageFromFile("boy2.png")
	if err != nil {fmt.Println("Error loading player image:", err)}
	var mapAsIntSlice [][]int = makeCollideMap(gameMap)
	oneLevelGame := mapGame{
		Level:        gameMap,
		tileHash:     ebitenImageMap,
		drawableLevel: ebiten.NewImage(windowWidth, windowHeight),
		collider:     tilecollider.NewCollider(mapAsIntSlice, gameMap.TileWidth, gameMap.TileHeight),
		demoPlayer:    player{x: 200, y: 150, pict: playerPict},
	}
	buildDrawableLevel(&oneLevelGame)
	err = ebiten.RunGame(&oneLevelGame)
	if err != nil {	fmt.Println("Couldn't run game:", err)}
}
```

Player Input

- After building the map and updated main in previous two
  - Fairly standard player input check

```
func getPlayerInput() (dX, dY float64) {
	if ebiten.IsKeyPressed(ebiten.KeyW) ||
ebiten.IsKeyPressed(ebiten.KeyUp) {
		dY = -3
	} else if ebiten.IsKeyPressed(ebiten.KeyDown) ||
ebiten.IsKeyPressed(ebiten.KeyS) {
		dY = 3
	}
	if ebiten.IsKeyPressed(ebiten.KeyA) ||
ebiten.IsKeyPressed(ebiten.KeyLeft) {
		dX = -3
	} else if ebiten.IsKeyPressed(ebiten.KeyD) ||
ebiten.IsKeyPressed(ebiten.KeyRight) {
		dX = 3
	}
	return dX, dY
}
```

Easy update to draw

- Draw just add drawing player
  - highlighted

```
func (game mapGame) Draw(screen *ebiten.Image) {
	drawOptions := ebiten.DrawImageOptions{}
	screen.DrawImage(game.drawableLevel, &drawOptions)
	drawOptions.GeoM.Reset()
	drawOptions.GeoM.Translate(float64(game.demoPlayer.x),
float64(game.demoPlayer.y))
	screen.DrawImage(game.demoPlayer.pict, &drawOptions)
}
```

update

- Update used to do nothing
  - Now checks for collisions and possibly removes player movement

```
func (m *mapGame) Update() error {
	Player_dx, Player_dy := getPlayerInput()
	final_dx, final_dy := m.collider.Collide(m.demoPlayer.x, m.demoPlayer.y,
		float64(m.demoPlayer.pict.Bounds().Dx()),
float64(m.demoPlayer.pict.Bounds().Dy()),
		Player_dx, Player_dy, nil) //the final nil is a callback that we could have
called if a collision occurs
	m.demoPlayer.x += final_dx
	m.demoPlayer.y += final_dy
	return nil
}
```

See the whole thing

- Let’s take a look at the whole thing
  - https://github.com/jsantore/SimpleCollideDemo
  - We have reached the level where the ‘simplest stripped down demo’ isn’t quite as simple any more.