HOMEWORK 05

Bubble Bobble – Mode 0 Game

Purpose: To build a Mode 0 game to further your understanding of tile modes, sprites, and backgrounds.

Instructions

For this homework, you will recreate the classic 1986 arcade game, ***Bubble Bobble***, in **Mode 0**. To get a better feel for the game, you can see some play-throughs on Youtube, here and here for example. ***Outside of the code in the files provided in the Mode 0 scaffold, you must write all code yourself. This includes not copying from previous lab code.***

You are free to expand upon or add your own twist to the game, but the core components outlined in the requirements must be present. If you decide to do this, *please speak to Aaron or a TA first* so we can help you determine whether it still fits the requirements.

You are welcome to use tilesets and tilemaps sources from the internet for your game state background.

Requirements:

Minimum Game Features

Your *game* must have the following:

- **Player Dragon (Bub)** that can move in the game. The goal is to defeat all enemies and get as high a score from food items as possible, without losing all lives.
    - The player should be pulled downward at a constant velocity unless they are supported by a platform.
    - The player should be an animated sprite with states of animation for a move left, a move right, and a jump with the left and right state having 3 frames of animation.
        - Movements jump, left, and right should each have a state with different appearance.
    - The player should be able to use their attack (blowing bubbles) by pressing a button. This bubble should visibly float up the screen the screen (hint: use a sprite) until colliding with an enemy or a wall, or despawning after a certain period of time.
    - Whenever the player loses a life, they should respawn at their initial location.
    - *Note:* It does not need to look exactly like the original version! Feel free to create your own sprite with its own original design for this.
- Player is able to **press the A button to blow a bubble**. Colliding with one of the active bubbles makes enemies unable to move for a short time, and able to be killed by the player.
    - There must be a visual representation of the bubble.
    - The bubble should disappear after some time.
    - The bubble should appear right in front of the player when deployed.
- At least **six enemy monsters** that are chasing the player, per-level.
    - They should each be an animated sprite with different states of animation for movement left, right, and in a bubble, and 2 frames of animation each.
    - The enemies should have a different appearance than the player.
    - At the beginning of the game, the enemies should be scattered throughout the map, and then start chasing the player.
    - When one of the enemies collides with the player, they should lose a life.
    - Check out the Tips section for a very simple algorithm to implement chasing.
    - When enemies are killed by the player, they should spawn a food at their location.

    - There should be at least two types of enemies with different appearances, chosen from the following list:
        - Type 1: Just walks/jumps around like the player can.
        - Type 2: shoots fireballs that just go straight until they hit a wall (can also jump)
        - Type 3: bounces off the level and moves like my screensaver from lecture ~3 (no jumping, no shooting, just flying)
        - Type 4: moves back and forth (no jumping), but shoots bullets downward like a space invader
        - Type 5: Ghost that can move straight through all terrain to kill the player if they're taking too long with the level.
- **Food** spawns when enemies are defeated.
    - When the player collides with food, they should gain score, and the food should play an animation before disappearing.
- A **score** displayed on the screen.
    - The score should increment each time the player collects food.
    - It should accumulate between levels.
        - e.g. if the player has a score of 1024 at the end of the first level, they should start the second level with 1024 and continue from there.
- A **lives tracker** displayed on the screen.
    - The player should start with 3 lives.
    - The tracker should be updated whenever the player is hit by an enemy monster.
- **Transparency** used in a majority of the sprites.
- At least **two levels**.
    - Each of these should have a different layout (tilemap and collision map).
    - Note: This is not exactly the same as how the game is sometimes implemented, with scrolling between levels. We specifically want you to DMA in a new background between levels.
- A **state machine** with the following states: **Start, Pause, Game1, Game2, Win, and Lose.**
    - *Win condition:* player defeats all enemies in both levels and eats the subsequent food.
    - *Lose condition:* player loses all lives.
- Uses a **tileset** to construct a tilemap for the Game states.
    - Check out the Tilemap Tutorial section for instructions on how to do this!
    - You do *not* need to use Tiled, but it is the TA-approved way to go about it.
    - ***Do not modify the provided tileset, already loaded into character block 0.***

- Uses a **collision map** in the Game state to detect collisions between the monsters and dragons, and the map.
    - I recommend waiting to implement this until we cover it!
    - The player should be able to fall through the bottom of the map back up to the top of the map.

Code / Files

Your *code* must have the following:

- Be entirely written in Mode 0.
- Good organization.
- Meaningful comments.
- A README.md file.
    - An instruction manual that tells a player how to play your game, like you implemented for previous homeworks.
    - Include here the things you did for extra credit, or anything else you'd like us to take into consideration when grading!

Extra Credit

Game mechanics that make your game more like the original could earn extra credit. If you decide to do this, *please outline what you implemented in your README file*.

Some examples of this include a second player (Bob), implementing more types of enemies, or implementing an enemy boss.

Tips

- Start early. Never underestimate how long it takes to make a game!
- The simplest way to determine enemy movement is to compare its position to the player, and then move it towards that location.
    - For example, if the player is at (30, 0), and the enemy is at (20, 20), the enemy "wants" to increment its x position, and decrement its y position.

Submission Instructions

Ensure that **cleaning** and building/running your project still gives the expected results. **Please reference the last page of this document for instructions on how to perform a "clean" command.**

Zip up your entire project folder, including all source files, the Makefile, and everything

produced during compilation **(including the .gba file)**. Submit this zip on Canvas. Name your submission
HW05_LastnameFirstname. For example:

> “HW05_BlubbaBarbara.zip”

It is *your* responsibility to ensure that:

- The zip file is named correctly.
- All the appropriate files have been submitted.
- Your submitted zip can be opened and everything cleans, builds, and runs.

Tilemap Tutorial

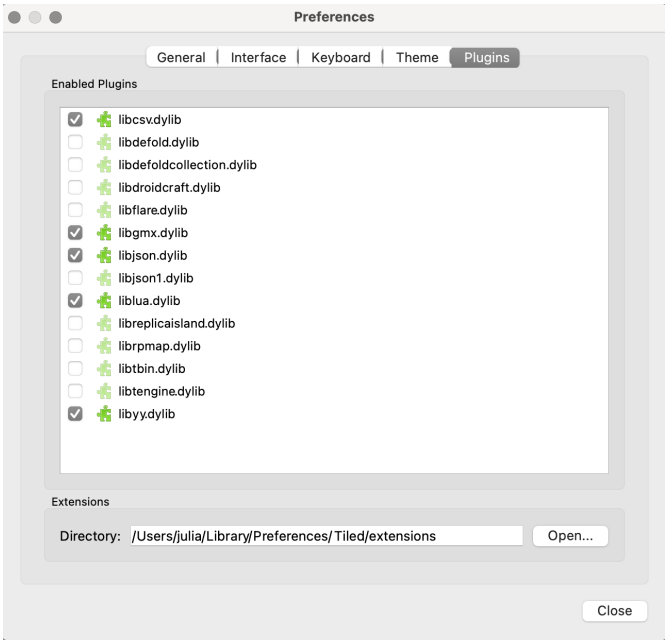
In this tutorial, we're going to be using Tiled to export the contents of our tilemap array. You're allowed to use another software for this, but keep in mind that you must use the exact tileset that we're providing and loading into memory for you. The only exception to this is changing the palette.

If you don't have Tiled installed yet, download it at this point!

Plugin installation

Unzip the HW05 Scaffold. You should find tiled-to-gba-export.js in there. This is the 2261 version of a plugin for Tiled (modified from source), allowing us to export in a useful format for tiled modes on the GBA.

Open Tiled, and go to Preferences > Plugins. Then, click on "open" next to the extensions directory.

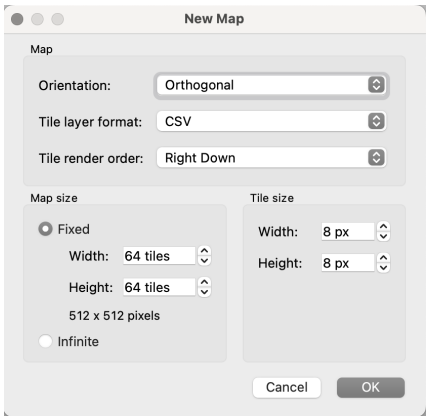


This should open a new Finder window at that location. Move your tiled-to-gba-export.js here. This extension will allow us to export our tilemap in chunks the size of screenblocks, so that our large background is already in the correct layout for how VRAM is mapped in tiled modes (such as Mode 0).

Building tilemap

You should also find tileset.png and tileset.tx in your unzipped folder. In the home page of Tiled, click on "open a file or project" and select tileset.tx.

You should now see the image representation of that tileset. Go to File > New > New Map (or press cmd/ctrl + N) to create a new tilemap. Your settings for the tilemap should look like this:



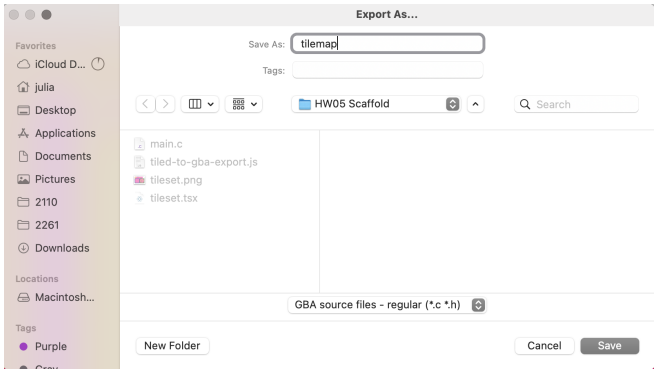
Click OK to create the map. You should see the tiles on the right of the screen. Now,

you can use those tiles to create your tilemap. Select a tile and start painting with it! Tiled has a lot of useful tools and features to effectively create tilemaps.

Note: Flipping (horizontal or vertical) should be supported by the extension, but rotation is not (since flipping tiles is a GBA feature). The tileset includes all flips/rotations of the important tiles. This means that you shouldn't have to use the flips in Tiled at all, but are welcome to.

Exporting tilemap

Once your tilemap is ready, go to File > Export As… and export it as GBA source files. It should look like this:



If you don't see the GBA source files option, try repeating the plugin installation steps.

At this point, you should have a `tilemap.c` and `tilemap.h` in your HW05 Scaffold folder. Check it out – it should hold an array called `[filename]Map`, with each of the tilemap indices separated in screenblock chunks.

Loading tilemap into memory

In main.c, you can see that we left a TODO for you to load in your tilemap. Use DMA to do this! Don't forget to `#include` the .h file that Tiled created for you.

There's also the `testTilemap` function. We wrote this for you to check if your tilemap has been created and loaded into the appropriate place in memory. If that is the case, you should be able to move around in the map (without going past the boundaries/seeing any wrapping) to see its entire contents.