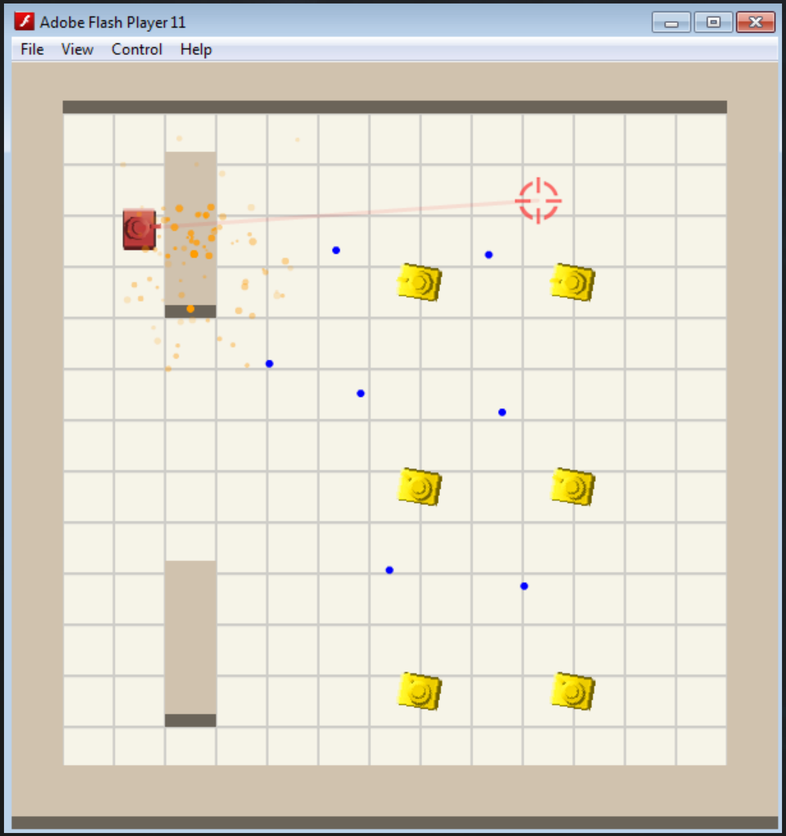
GAME CREATION CLUB - TUTORIAL

Tile Based Game using FlashDevelop -Part 1-

GAME CREATION CLUB - TUTORIAL

Overview – Part 1



- Flash/ActionScript 3
- FlashDevelop
- Tile Based Engine
- Game Components Structure
- Rendering Tiles from 2D Array

Flash / ActionScript 3

- Flash is multimedia, software platform that can be played and executed in Adobe Flash Player (wikipedia)
- Games on newgrounds and armorgames are most likely made in Flash
- Most GCC games are made using Flash
- ActionScript 3 is an object-oriented language with similarities to javascript and java

FlashDevelop

- Free, open-source code editor for developing Flash applications

- Pros (compared to Flash CS6)
  + Free (Flash Professional CS6 is $185 on amazon)
  + Similar to other IDE (eclipse, visual studio, etc.)

- Cons
  - Lack of features (sprite sheet generation, timeline, etc.)

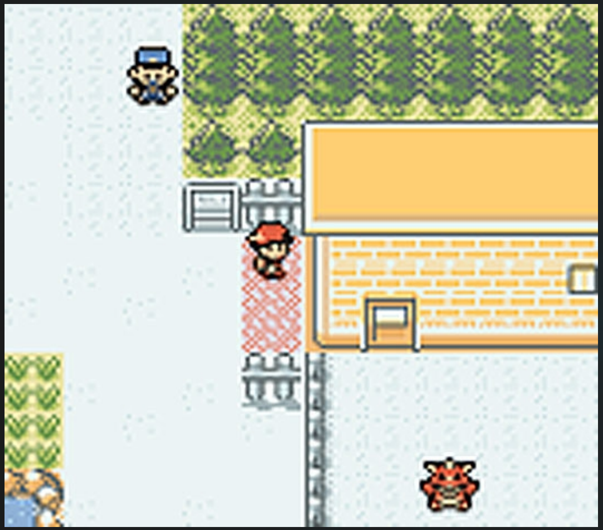
Tile Based Game Engine

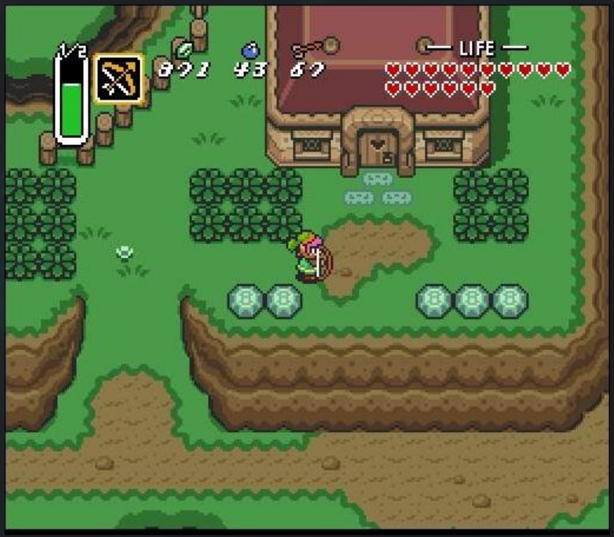
- Tile Based Video Game is a form of video game where the playing area is generated by laying out tiles adjacent to one another in a grid (wikipedia)

- Pros:
  + reusable assets
  + easy level creation
  + simplifies collision detection

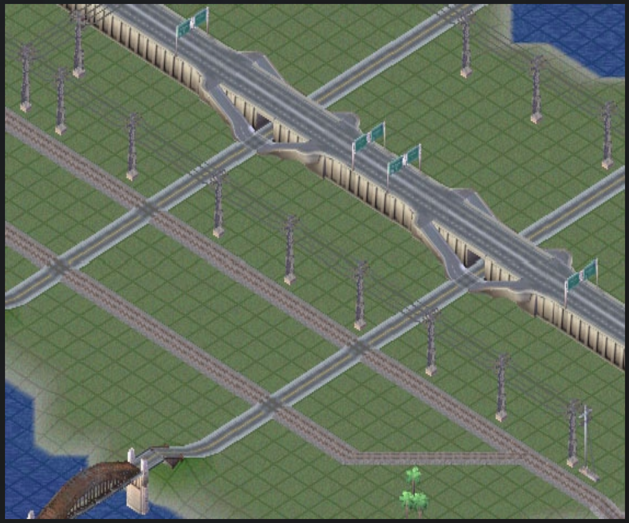
- Cons:
  - object placement is usually restricted to grids
  - outdated (probably won't see many console tile-based games anymore like SNES)

Commercial Tile Based Games


Pokemon



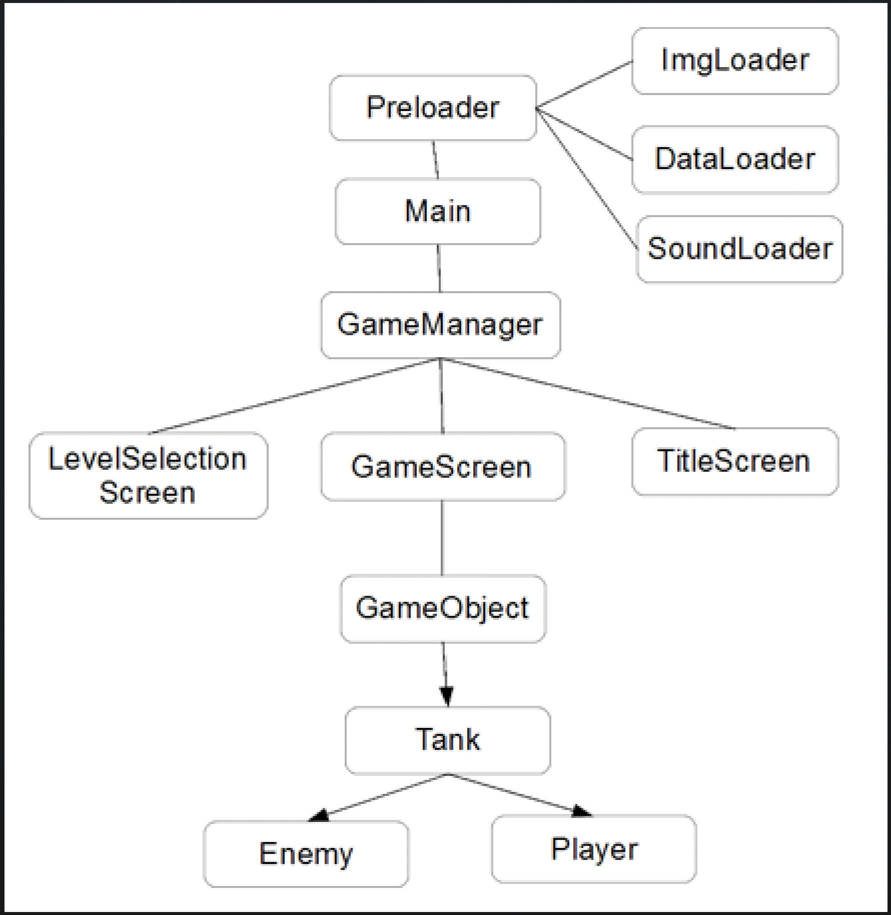
Old School Zelda


Sim City 3000

Tile Based Tanks

- Download the template from here: https://www.dropbox.com/sh/r9jcsrkexeytye4/XyDSAyKzgn
- Download “TileBasedTanks_pt1.zip”
- After the tutorial, download “TileBasedTanks_pt1_done.zip” to make sure I didn't make any mistakes in the slides (sorry I'm quite mistake prone)

Program Structure of Tile Based Tanks



- In today's tutorial, we will be filling in codes for Main, GameManager, and GameScreen
- Don't worry about the preloader since I have already implemented that

Adding main game loop to stage

- From the right panel, open “src/Main.as”
- Then in “private function init”, add this code:

```
stage.addEventListener(Event.ENTER_FRAME,loop);
```

- The function that is passed as an argument (loop) will be called each frame. Since our game will run at 30 fps, loop() will be called 30 times per second

- Now under function init(), create a new funtion:

```
private function loop(e:Event):void {

}
```

Creating BitmapData and Bitmap

- In “Game/GameManager.as”, create a new instance variable: private var gameScreen:GameScreen;
- This will create an instance variable of type GameScreen

- Then add two more instance variables:

```
private var Renderer:BitmapData;
private var bitmap:Bitmap;
```

- Then instantiate the instance variables in “public function GameManager”:

```
Renderer = new BitmapData(stageW, stageH, false, 0x000000);
bitmap = new Bitmap(Renderer);
gameScreen = new GameScreen(stageW, stageH);
```

Blitting

- By using bitmapData and bitmap, we are using a rendering technique called blitting, where we manipulate single bitmapData to render assets to stage. So we draw the image in memory first before actually drawing to stage.

- Now to actually manipulate the bitmapData, in function Render():

```
private function Render():void {
    Renderer.lock();
    Renderer.fillRect(Renderer.rect, 0xFFFFFF);
    gameScreen.Render(Renderer);
    Renderer.unlock();
}
```

Adding the bitmap to stage

- Now return to "main.as" and create an instance of GameManager:

```
private var gm:GameManager;
```

- Then instantiate it in init():

  *** changed from stage.stageWidth, stage.stageHeight to 600,600 (google chrome changes stage size for some reason when I do stageWidth and stageHeight) ***

```
gm = new GameManager(600, 600);
addChild(gm.bitmap);
```

- addChild(gm.bitmap) adds the bitmap to stage. It won't show up on stage unless we do so.
- Now in loop(), add:

```
gm.Render();
```

GameScreen

- This class will handle the actual gameplay
- In “Game/Screen/GameScreen.as”, create instance variables:

```
private var tiles:Array;
private var tileSize:int = 40;
private var depth:int = 5;
private var currentLevel:int = 1;
```

- Now under function GameScreen():

```
tiles = DataAccessor.getStage(currentLevel).tiles;
```

DataAccesor

- This is a class that I created to access data loaded from data.json, where all the information about tanks and stages are located
- DataAccessor.getStage(id) will give you access to stage with id of id. DataAccessor.getStage(id).tiles will give you access to the tile data of the stage
- You can manipulate tanks and stages by editing “bin/lib/data/data.json”

Rendering tiles from tiles Array - 1

- * change the parameter of Render() in GameScreen.as to Render(Renderer:BitmapData) *
- Now under function Render() in GameScreen.as:

```
var m:Matrix = new Matrix();

for(var i:int = 0; i< tiles.length; i++){
        for(var j:int =0; j < tiles[0].length; j++) {

                // FILL IN LATER
        }
        m.tx = 0;
        m.ty += tileSize;
}
```

- The nested for loops will access each entry in tiles Array

Rendering tiles from tiles Array – 2

- We will now fill in //FILL IN LATER
- In the for loop: for(int j:0; j < tiles[0].length; j++):

```
    if(tiles[i][j] >= 0 && tiles[i][j] < 10) {
        Renderer.draw(ImgAccessor.getImg("floor_tiles", tiles[i][j]), m);
    }
    else if(tiles[i][j] >= 10 && tiles[i][j] < 20) {
        m.ty -= depth;

        // I added a ) after 10
        Renderer.draw(ImgAccessor.getImg("solid_tiles",tiles[i][j]-10),m);
        m.ty += depth;
    }
    m.tx += tileSize; // sorry I forgot this
```

I think it works now

- Now go ahead and click on the blue triangle next to “Debug” at top
- After an ugly now loading screen, you will see the two dimensional array stored in data.json rendered on the screen
- To manipulate the stage data, go to data.json and under “stages” , go to the stage object with “id”: 1, and you can change the tiles array in “tiles”
- When you change “00” to “10”, you will see a solid tile rendered at that position corresponding to the tile array
- Next week (probably), I will be going over rendering a player tank on to the screen and controlling it, and collision detection between our player tank and the rendered tiles.