Phaser Game Engine

TAMZ 1
Lab 8

See: Lecture 6 & referenced materials

https://phaser.io/

Phaser 3.3.0 "Tetsuo" is the current stable version and was released on 22nd March 2018.

PHAS3R

🛸 PHASER FEATURES

WEBGL & CANVAS	INPUT
PRELOADER	SOUND
PHYSICS	TILEMAPS
SPRITES	DEVICE SCALING
GROUPS	PLUGIN SYSTEM
ANIMATION	MOBILE BROWSER
PARTICLES	DEVELOPER SUPPORT
CAMERA	WEB FIRST

Downloading Phaser

Phaser 3 is available via GitHub, npm and CDNs

- <script src="//cdn.jsdelivr.net/npm/phaser@3.16.2/dist/phaser.js"></script>
- <script src="//cdn.jsdelivr.net/npm/phaser@3.16.2/dist/phaser.min.js"></script>

Dowload js:
- https://phaser.io/download/stable

Github:
- https://github.com/photonstorm/phaser/tree/v3.3.0

Npm
- npm install phaser@3.3.0

Using Phaser

Phaser 3 API Documentation can currently be found here:
- https://phaser.io/phaser3/api/components

Examples:

- http://labs.phaser.io/index.html
- https://github.com/photonstorm/phaser3-examples
- https://phaser.io/tutorials/making-your-first-phaser-3-game
- https://stackoverflow.com/questions/tagged/phaser-framework

Phaser Authors' Note: Not all examples work, sorry! We're tidying them up as fast as we can.

Configuring Phaser

config object: configure your Phaser Game
- WebGL or Canvas
- game width
- game height
- created scene

game object: instance of a Phaser.Game
- the configuration object is passed to it

life-cycle:
- init
- preload
- create
- update

```
var config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};

var game = new Phaser.Game(config);

function preload ()
{
}

function create ()
{
}

function update ()
{
}
```

Preloading Files

- Phaser will automatically look for this function when it starts and load anything defined within it - Phaser loads images and assets into memory before launching the game

- this example will load in 4 assets (1x image, 2x spritesheets, 1x tilemap)

- the first parameter represents the **asset key** - link to the loaded asset

```javascript
function preload ()
{
    this.load.spritesheet('robot', 'assets/lego.png',
        { frameWidth: 37, frameHeight: 48 } );

    this.load.spritesheet('items', 'assets/items.png',
        { frameWidth: 32, frameHeight: 32 } );

    this.load.image('tiles', 'assets/map_tiles.png');
    this.load.tilemapTiledJSON('json_map', 'assets/json_map.json');
}
```

Creating Player Object

- after preloading phase, the **create** method is executed - place to create entities for your game (player, enemies, etc)

- **asset key** – is used when creating Game Objects

- the values 300 and 450 are the x and y coordinates of the sprite

```
function preload ()
{
    this.load.spritesheet('robot', 'assets/lego.png',
        { frameWidth: 37, frameHeight: 48 } );

    this.load.spritesheet('items', 'assets/items.png',
        { frameWidth: 32, frameHeight: 32 } );

    this.load.image('tiles', 'assets/map_tiles.png');
    this.load.tilemapTiledJSON('json_map', 'assets/json_map.json');
}

function create()
{
    player = this.add.sprite(300, 450, 'robot');
}
```

Using (Arcade) Physics

```javascript
function create()
{
    player = this.physics.add.sprite(300, 450, 'robot');
    cursors = this.input.keyboard.createCursorKeys();

    this.anims.create({
        key: 'run',
        frames: this.anims.generateFrameNumbers('robot', { start: 0, end: 16 }),
        frameRate: 20,
        repeat: -1
    });

}

function update ()
{
    player.anims.play('run', true);

    if (cursors.left.isDown)
    {
        player.body.setVelocityX(-150);
        player.angle = 90;
    }
}
```

```javascript
var config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: { y: 300 },
            debug: false
        }
    },
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};
```

- scene is running - the **update** method is executed multiple times per second
- to use **arcade physics** system, we need to add it to our game config

Setting up Animations

```
function create()
{
    player = this.physics.add.sprite(300, 450, 'robot');
    cursors = this.input.keyboard.createCursorKeys();

    this.anims.create({
        key: 'run',
        frames: this.anims.generateFrameNumbers('robot', { start: 0, end: 16 }),
        frameRate: 20,
        repeat: -1
    });

}

function update ()
{
    player.anims.play('run', true);

    if (cursors.left.isDown)
    {
        player.body.setVelocityX(-150);
        player.angle = 90;
    }
}
```
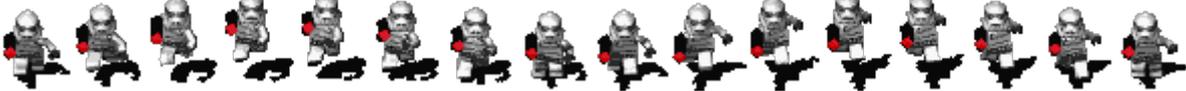
- Animation –
- Robot is loaded (in **preload** function) as a sprite sheet – becauseit contains animation frames
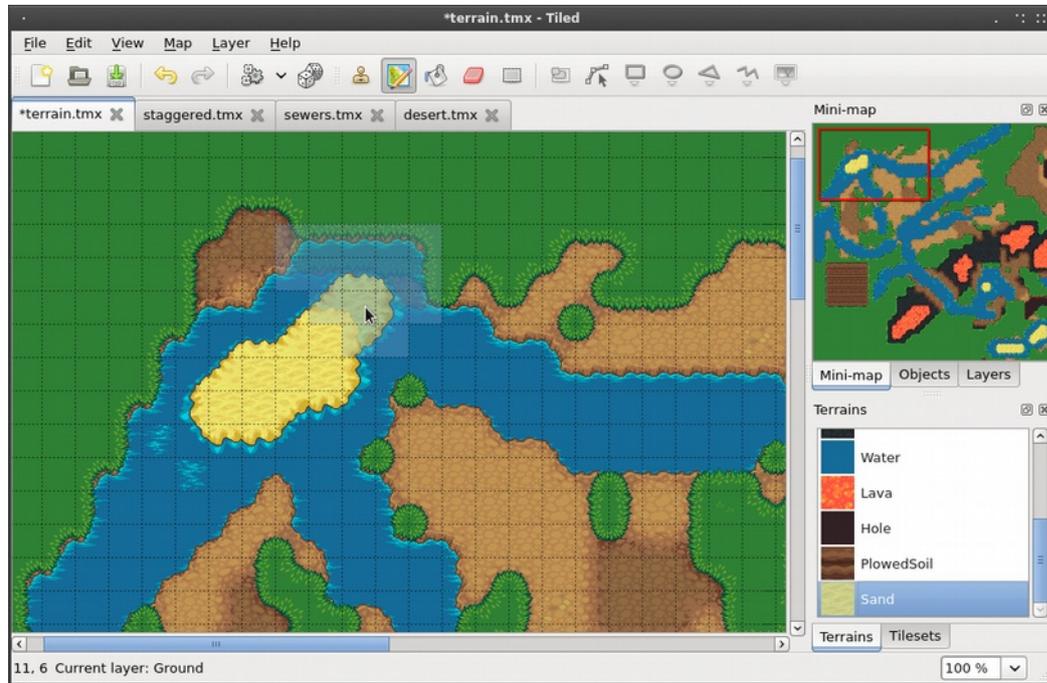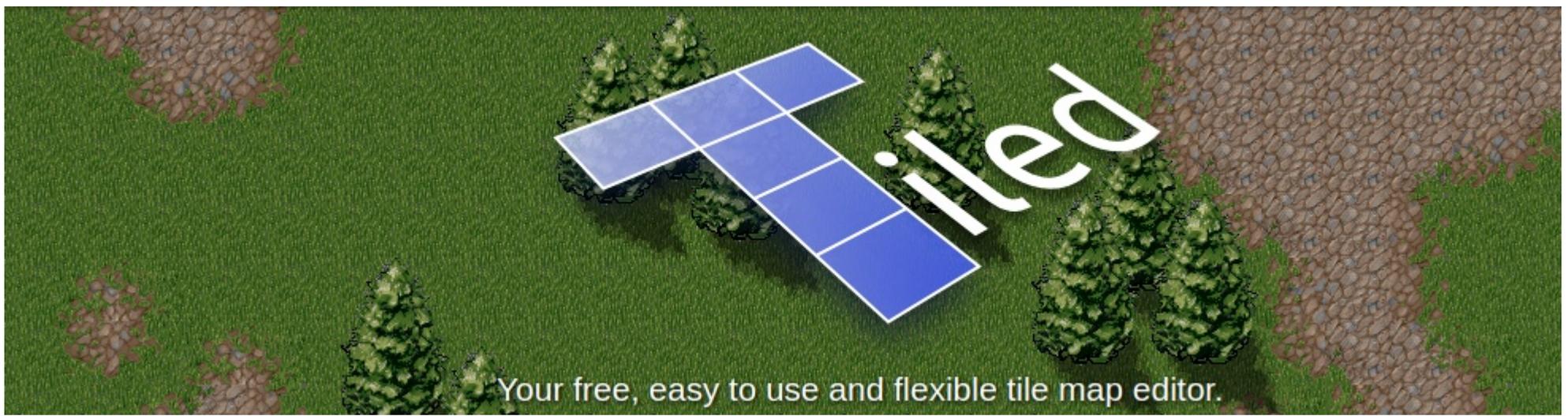
http://www.mapeditor.org/ "Tiled is a 2D level editor that helps you develop the content of your game. Its primary feature is to edit tile maps of various forms, but it also supports free image placement as well as powerful ways to annotate your level with extra information used by the game. Tiled focuses on general flexibility while trying to stay intuitive." - http://doc.mapeditor.org/en/latest/manual/introduction/#about-tiled
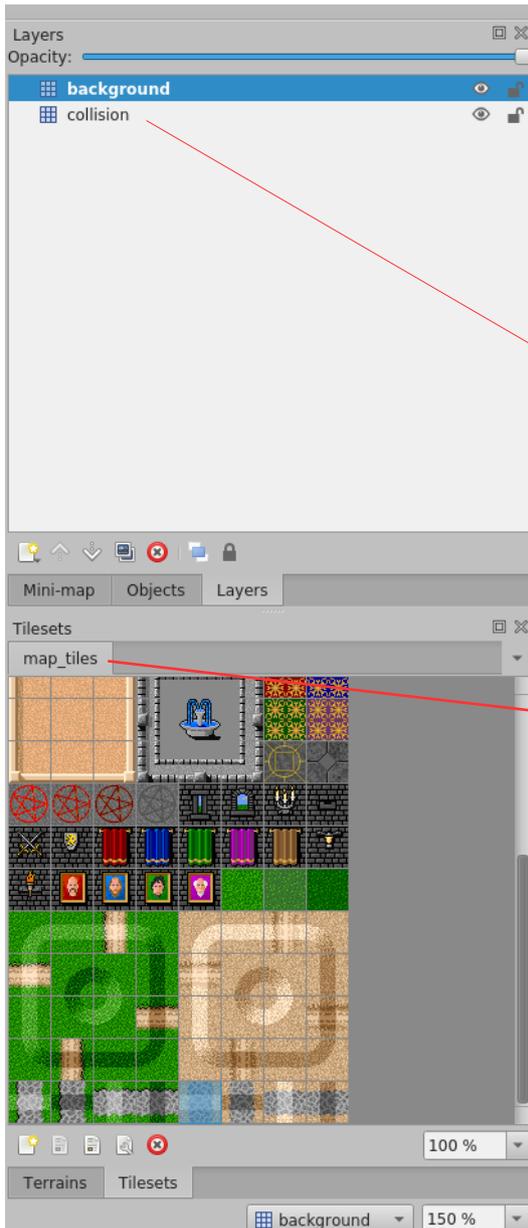
Adding a Map to the Game


```
function preload ()
{
    this.load.spritesheet('robot', 'assets/lego.png',
        { frameWidth: 37, frameHeight: 48 } );

    this.load.spritesheet('items', 'assets/items.png',
        { frameWidth: 32, frameHeight: 32 } );

    this.load.image('tiles', 'assets/map_tiles.png');
    this.load.tilemapTiledJSON('json_map', 'assets/json_map.json');
}

function create()
{
    map = this.make.tilemap({ key: 'json_map' });
    //'map_tiles' - name of the tilesets in json_map.json
    //'tiles' - name of the image in load.images()
    var tiles = map.addTilesetImage('map_tiles','tiles');

    backgroundLayer = map.createDynamicLayer('background', tiles, 0, 0);
    collisionLayer = map.createDynamicLayer('collision', tiles, 0, 0);

}
```

We can create a map with static/dynamic layers from a JSON file

https://labs.phaser.io/index.html?dir=game%20objects/tilemap/

Controlling Movement

```
function create()
{
    map = this.make.tilemap({ key: 'json_map' });
    var tiles = map.addTilesetImage('map_tiles','tiles');
    backgroundLayer = map.createDynamicLayer('background', tiles, 0, 0);
    collisionLayer = map.createDynamicLayer('collision', tiles, 0, 0);

    this.anims.create({
        key: 'run',
        frames: this.anims.generateFrameNumbers('robot', { start: 0, end: 16 }),
        frameRate: 20,
        repeat: -1 });

    player = this.physics.add.sprite(300, 450, 'robot');
    cursors = this.input.keyboard.createCursorKeys();

}
function update ()
{
    // left movement
    if (cursors.left.isDown)
    {
        player.anims.play('run', true);
        player.body.setVelocityX(-150);
        player.angle = 90;

    }
}
```

- Phaser has a built-in Keyboard manager
  - properties: up, down, left, right
- if the left key is being held down → negative horizontal velocity + animation

Detecting Object Collisions

Collision example (experiment with the options):

```
//Sets collision on all tiles in the given layer,
//except for the IDs of those in the given array
collisionLayer.setCollisionByExclusion([ -1 ]);
//collisionLayer.setCollision();
//collisionLayer.setCollisionByIndex();
//collisionLayer.setCollisionBetween();
// We want the player to physically collide with the collisionLayer,
// but the backgroundLayer layer should only trigger an overlap
this.physics.add.collider(player, collisionLayer);
this.physics.add.overlap(player, backgroundLayer);

//Checks to see if the player overlaps with any of the items,
//if he does call the collisionHandler function
this.physics.add.overlap(player, items, collisionHandler);


function collisionHandler (player, item) {
    updateText();
    item.disableBody(true, true);
    if (item.body.enable == false)
    {
        //new random location
        item.setFrame(itemID);
        item.enableBody(true, itemX, itemY, true, true);
    }
}
```

Assignment

Task (1-2pt)

- During the lab, we will show, how to move the player, center the screen on it and detect collisions with walls.
- Continue with the game example.
  - Experiment with the mapeditor, player animation, collision system, etc.
- The task is to make the player collect items which will be randomly displayed over the map and show actual score (which will be reset if player does not collect an item before timeout)
- Additional point will be awarded if the player has several enemies (bouncing bombs) which it has to avoid (otherwise the score is reset)