

LUDUM

How To Use Game Engine

March 7, 2019

Contents

1	Overview	5
1.1	Install Ludum game engine	5
1.1.1	Requirements	5
1.1.2	Installation	5
2	Components	7
2.1	Entity	7
2.2	Body	8
2.3	CollisionDetection	9
2.4	Physics	9
2.5	AudioManager	10
2.6	Sprite	10
2.7	ImageRender	10
2.8	ResourceManager	11
2.9	Background	11
2.10	HUD	12
2.11	Menu	12
2.12	ScoreBoard	12
2.13	Logger	12
3	More Information	15
3.1	Contact	15

Chapter 1

Overview

1.1 Install Ludum game engine

1.1.1 Requirements

- vscode
- nodejs
- npm or yarn
- jest

1.1.2 Installation

Open terminal in vscode. Go to project folder via terminal and type:

- npm i
- npm start

Chapter 2

Components

2.1 Entity

All objects that the user wants to create get created via the Entity class.

The constructor of Entity takes these arguments:

- name
- body
- physics
- collisionDetection
- audioManager
- sprite
- imageRender

Example of how to create a *Entity*:

```
1 class Box {  
2     constructor(x, y, height, width) {  
3         this.entity = new Entity(  
4             "Ludum",  
5             new Body(this, x, y, height, width),  
6             new Physics(this, 10, -6),  
7             new CollisionDetection(this),  
8             null,  
9             null,  
10            new ImageRender(this, ResourceManager.getImagePath("logo.  
11            png"))  
            );
```

```
12 }  
13 }
```

Entity has these methods. These methods return either the component or null.

- `getEntity()` returns entity
- `getName()` returns name of entity as a string
- `getBody()` returns body
- `getPhysics()` return physics
- `getCollisionDetection()` return `CollisionDetection`
- `getAudioManager()` returns `audioManager`
- `getSprite()` returns sprite
- `getImgae()` returns image
- `getUpdate()` updates body of entity through physics
- `getEntityProps()` returns the newest value in body and physics

2.2 Body

Body class is the body of the entity.

The constructor of `Body` takes these arguments:

- `entity`
- `left`
- `top`
- `height`
- `width`

The body class contains only setters and getters for these parameters.


```

1 class Object {
2   constructor() {
3     this.entity = new Entity(
4       "Object",
5       new Body(this, 300, 540, 100, 100),
6     )
7   }

```

Here is a small example of how to move the entity bird:

```

1
2 if (this.getBody().getTop() > 1040) {
3   this.getBody().setTop(400);
4   this.getBody().setLeft(300);
5 }

```

2.3 CollisionDetection

To check for collisionDetection use:

```

1 checkForCollision(otherEntity)

```

Example of this can be:

```

1
2 let hasPlayerCollided = player
3   .getCollisionDetection()
4   .checkForCollision(this.state.playerArr[index].
   getEntity());

```

2.4 Physics

Physics takes 3 arguments:

- entity
- left
- top

Physics has setters for these items, while getters for left and top.

Physics also has a update arrow function. This arrow function is used to update the physics of the object from old to new position.

This method is used in Entity update.

Example:

```

1 placeholder[0].getPhysics()
2 .setTop(this.state.playerArr[0].getPhysics().getTop() * -1);

```

This example is about changing the vertical direction of an object with physics.

2.5 AudioManager

example to use AudioManager:

Object:

```

1 new AudioManager([
2     ResMan.getAudioPath("soundEffect1.mp3"),
3     ResMan.getAudioPath("soundEffect2.mp3"),
4     ResMan.getAudioPath("soundEffect3.mp3")
5 ],
6 this.enum = {
7     BIRD_JUMPS: 0,
8     BIRD_SCORE: 1,
9     BIRD_DIES: 2
10 });

```

Then if something happens:

```

1
2 object.getAudioManager().play(2); // testing!!!

```

2.6 Sprite

2.7 ImageRender

To use ImageRender, simply add:

```

1 new ImageRender(this, ResourceManager.getImagePath("logo.png"))

```

To an entity.

Example:

```

1 class Box {
2     constructor(x, y, height, width) {
3         this.entity = new Entity(
4             "Ludum",
5             new Body(this, x, y, height, width),
6             new Physics(this, 10, -6),
7             new CollisionDetection(this),
8             null,
9             null,

```

```

10     new ImageRender(this, ResourceManager.getImagePath("logo.
11     png"))
12 );

```

To get this to work, you need to add:

```

1  getImage() {
2      return this.entity.getImage();
3  }
4  // rendering this class
5  render() {
6      return <span className="frame">{this.getImage().render()}</
7      span>;
8  }

```

To the object file.

2.8 ResourceManager

To use the ResourceManager simply import the class and use it like this:

```

1 ResourceManager.getImagePath("background.png")
1 ResourceManager.getAudioPath("one.mp3")
1 ResourceManager.getSpritePath("birds.png")

```

The *ResourceManager* uses a default paths:

- ../resources/image/
- ../resources/audio/
- ../resources/sprite/

You can change these paths in the *resourceManager* file

2.9 Background

To use the *Background* component, simply add it to the component.

```

1 <Background
2     height={1080}
3     width={1920}
4     speed={0.5}

```

```

5         image={ResourceManager.getImagePath("background.png
      ")}
6       >
7         {" "}
8     </Background>{" "}

```

ackground contains *defaultProps*, so it is not needed to set *height*, *width* and *speed*. To set a image you can either use the *ResourceManager* or simply import a image.

2.10 HUD

To use the HUD simply add the HUD component

```

1 <HUD score={this.state.score} position={"tc"} />{" "}

```

Where *score* is a score variable from the game.

2.11 Menu

To use the menu, simply import the component into the file you want.

```

1 <Menu showMenu={this.state.showMenu}

```

Using *this.state.showMenu* gives you the option of toggling it on and off, depending of a boolean *showMenu*.

To add more menu options, simply add more items into the *menuItems*, and use *handleClick(e)* for the event.

2.12 ScoreBoard

To use scoreboard simply add:

```

1 <ScoreBoard />

```

To get the score from the game, *context* is recommended, as it is shown in the *flappy* demo, since normally a *menuItem* shows a scoreboard.

2.13 Logger

To use the logger simply use:

```

1 Logger.setText("flappy.js", 'score: ${this.state.score}');

```

where first argument is the name of file and second argument is what you want to log.

Then you can add this to the game:

```
1 <LoggerManager />
```


Chapter 3

More Information

To learn more about the game engine visit our github projects:

github.com/bergurijohansen/LudumGameEngine

github.com/bergurijohansen/LudumFlappy

github.com/bergurijohansen/LudumDino

3.1 Contact

Contact information:

Herborg Kristoffersen

Bergur I. Johansen

Helgi Poulsen

Lív Olsen

Fróði H. Joensen