

**Gruppenarbeitsbericht**

Zum Thema:

Entwicklung eines Tower Defence Spiel mittels createjs Bibliothek

Studenten**:**

**DEIAN DIMITROV** 201314004

**VALERI VATCHEV** 201314026

**MARTIN GEBOV** 201314009

Sofia

2015

**Inhaltsverzeichnis**

[1. Einleitung 3](#_Toc423307911)

[1.1. Spieltypen 4](#_Toc423307912)

[1.2. Ziel des Arbeit 4](#_Toc423307913)

[2. Problemstellung 5](#_Toc423307914)

[2.1. Spiel Einheiten 5](#_Toc423307915)

[2.2. Grid 5](#_Toc423307916)

[2.3. IO Events 5](#_Toc423307917)

[3. Theoretische Lösung 6](#_Toc423307918)

[3.1. Grid 6](#_Toc423307919)

[3.2. Menüs 6](#_Toc423307920)

[3.3. Bewegung 6](#_Toc423307921)

[3.4. A\* Algorithmus 6](#_Toc423307922)

[3.5. Entities 6](#_Toc423307923)

[3.6. Enemies/Creeps 6](#_Toc423307924)

[3.7. Bosses 7](#_Toc423307925)

[3.8. Türme 7](#_Toc423307926)

[3.9. Waves 7](#_Toc423307927)

[3.10. User Interface 7](#_Toc423307928)

[3.11. Sound 8](#_Toc423307929)

[3.12. Animation 8](#_Toc423307930)

[4. Praktische Lösung 9](#_Toc423307931)

[4.1. CreateJS 9](#_Toc423307932)

[4.2. Entwicklung 9](#_Toc423307933)

[5. Zusammenfassung 20](#_Toc423307934)

[6. Literaturverzeichnis 21](#_Toc423307935)

# Einleitung

Als Gruppenprojekt wird ein Computersielt entwickelt. Die drei Studenten, die an dem Projekt teilnehmen, projektieren und entwickeln verschiedene Features für das Spiel. Es wurde ein GIT benutzt, um alle Studenten ein Angriff zu haben zu den neuste Version den Spieles.

Die Studenten haben entschieden ein Tower Defence Spiel zu entwickeln.

## Spieltypen

Das Spiel wird ein 2dimensionaler Modus aus der Tower Defence Genre. Tower Defence Spiele haben ungefähr den folgenden allgemeinen Regeln:

### Tower Defense

Bei der einfachsten Variante laufen die Gegner eine festgelegte Strecke ab. Dabei können keine Türme auf der Strecke gebaut werden, um zu verhindern, dass der Spieler den Weg blockiert. Dadurch wird es für den Spieler komplizierter, da er die Strecke nicht verändern kann und somit die Länge immer gleich bleibt, die verwendet werden kann, um die Gegner zu stoppen.

### Mazing TDs

Neben dem Tower Defense existiert auch eine Variante, bei der die Strecke nicht festgelegt ist und sich die Creeps, die Gegner, selber einen Weg suchen. Bei dieser Variante spricht man auch von Mazing. Die Spielfläche zunächst leer ist und der Pfad der Creeps erst durch ein vom Spieler mit den Türmen gebildetes Labyrinth entsteht. Da man hier theoretisch den Weg versperren könnte, gibt es generell zwei verschiedene Implementationen, dies zu verhindern. So wird mithilfe von Pathfinding überprüft, ob überhaupt ein Weg existiert und dementsprechend der Bau blockierender Türme verhindert oder den Gegnern ermöglicht bebaute Gebiete zu überqueren bis erneut ein gültiger Pfad entsteht. Auch ist es möglich, dass die Gegner bei blockiertem Weg die Türme selbstständig zerstören. Bei Mazing Tower Defenses muss neben dem effektiven Bau der Türme auch die Wegführung für die Creeps möglichst gekonnt

## Ziel des Arbeit

Das Spiel, das realisiert wird ist Mazing Tower Defence. Es wurde so entschieden, weil bei der Entwicklung die Studenten sich, mit viele neue für ihnen Technologien, kennen lernen.

Jeder Student wird an das Projekt an der Graphik, Audio, Logik arbeiten. So wird auch für die Studenten interessant zu sehen wie jeder Teil des Spieles funktioniert. Nach der Realisierung des Projekts die Studenten werden die Kenntnisse haben eine Anwendung von nichts zu starten.

Die Technologien die gewählt sind, sind aktuelle Bibliotheken die frei gegeben sind von den Entwickler. Die dienen in ein HTML5 Canvas ein JavaScript zu laufen. Die JavaScript Bibliotheken sind verantwortlich für den Grafischen Teil des Anwendung, sowie den Audio und Ressourcen Ladung.

Die benutzten Technologien werden in der Praktische Lösung dieses Berichts weiter erklärt.

# Problemstellung

## Spiel Einheiten

### Gegner

Die Gegner in Tower-Defense-Spielen werden *Creeps* genannt. Sie tauchen in Form von mehr oder weniger großen Gruppen an einem bestimmten Eintrittspunkt der Karte auf und versuchen, zum Ausgang der Karte zu gelangen. Gelingt dies einer gewissen Zahl von Gegnern, verliert der Spieler die Runde. Bei den meisten Tower-Defense-Spielen sind die Gegner abgesehen davon jedoch wehrlos und nicht in der Lage, die Verteidigungsanlagen des Spielers zu beschädigen oder zu zerstören.

Einzelne Gegner besitzen eine bestimmte Zahl an Trefferpunkten und eine bestimmte Geschwindigkeit. Daneben können sie noch über weitere Eigenschaften wie beispielsweise Resistenzen verfügen.

### Turmtypen

Türme haben in fast jeder Variante eine bestimmte Reichweite, erzielen einen bestimmten Schaden, haben eine Kadenz und verursachen einmalig Baukosten. Die Meisten können gegen Bezahlung in Form von Geld oder Erfahrungspunkten aufgerüstet werden, entweder zur Verbesserung einer Eigenschaft oder als generelles Upgrade. Die Türme sind meist nach ihrer Bewaffnung oder Eigenart benannt, welche dem Spielekontext angepasst ist. So gibt es schadenverursachende Türme, die einen oder mehrere Gegner angreifen, andere schaden allen Gegnern in einem bestimmten Umkreis, oder können bestimmte in anderer Hinsicht immune Gegner verletzen.

Zusätzlich stehen generell auch Türme mit speziellen Eigenschaften zur Verfügung die Gegner stoppen, verlangsamen, vergiften oder (seltener) die Gegner an eine andere Stelle auf der Karte versetzen.

## Grid

Vor alles kommt die Frage mit den Grid auf welche das Spiel laufen wird. Wird es den ganzen Entwicklung Platz nehmen oder nicht. Wie kann man es skalieren und so optimieren das es wenigen Ressourcen verbraucht und schnelle Durchsuchungen ausführt.

## IO Events

Diese Events werden durch die ganze Ausführung der Anwendung beobachtet. Z.B wenn ein „Esc“ gedrückt wird, wird den Spielt pausiert, und ein Popup wird was auszeigen. Beim Spielen, wird den Maustaste druck verantwortlich sein für die Stellung neue Turme.

# Theoretische Lösung

## Grid

Realisiert wird ein Gitter. Der Funktion der Entstehung soll maximal flexibel gemacht sein. Der Entwickler soll zu die Methode den Start punkt von den Grid geben. Auch was den höhe und weite von den einzige Zellen, sollen als Parameter gegeben werden. Die Funktion wird danach ein Objekt zurückliefen, der wird an die vorgegebene Position platziert. Jede Zelle an den Gitter wird auch ein Objekt sein welcher hält die Zusatz Eigenschaften des Teiles. Z.B Ob drauf ein Turm gibt. Der Teil hält auch die Information über den kontinuierlicher Anzahl von zählen.

## Menüs

Nach den spielflache wird nachgedacht wie den Laden von Bilder und andere Ressourcen implementiert wird. In dieses Teil werden auch den Menüs gemacht. So dass am Anfang des Spiels ein Menü vorkommt und fragt ob der Benutzer ein spiel startet will oder er will den ganze Anwendung beenden. Wen ein Spiel anfangt, wird den Menu unsichtbar-weg gemacht und den spiel soll anfangen.

## Bewegung

Die Bewegung wird auf den Grid stattfinden, Es wurde in der Logik Teil implementiert wie genau ein Spiel Entity sich auch den Grid bewegen. Nach alle Rechnungen im Hintergrund wird eine Bewegung Animation vorgestellt.

## A\* Algorithmus

Der Suchalgorithmus A\* wird benutzt um den ein Pfad zu finden. Er dient in der Informatik der Berechnung eines kürzesten Pfades zwischen zwei Knoten in einem Graphen mit positiven Kantengewichten. Der Algorithmus gilt als Verallgemeinerung und Erweiterung des Dijkstra-Algorithmus, in vielen Fällen kann aber umgekehrt A\* auch auf Dijkstra reduziert werden.

Im Gegensatz zu uninformierten Suchalgorithmen verwendet der A\*-Algorithmus eine Schätzfunktion (Heuristik), um zielgerichtet zu suchen und damit die Laufzeit zu verringern. Der Algorithmus ist optimal. Das heißt, dass immer eine optimale Lösung gefunden wird, falls eine existiert.

## Entities

Das ist der Haupt klasse die alle türme und Enemies vererben. Es halten die Information von Koordinaten und auch der Teil Nummer.

## Enemies/Creeps

Die Creeps kommen von link und sollen ganz rechts in der Mitte des Gitters gehen. Die werden im paare oder Gruppen laufen lassen. Die haben auch verschiedene Geschwindigkeit und HP. Wen der Creep durch das Spiel Feld ist, entnimm es ein Leben von den Spieler.

### Health

Eigenschaft für den Leben des Creep. Wen der Health 0 wird, bedeutet das der Creep von den spiel raus genommen muss.

### Points

Jede Creep wen gestorben gibt ingame-currency an der Spieler.

### Level

Wen den Level von den Creep ist grösser bedeutet es wird auch mehr in-game currency abgeben. Von den Level des Creeps hängt auch wieviel Health es haben wird.

## Bosses

Den stärksten Gegner sind den Bosses die vielmehr mal Health und Points für Ausgeben haben. Wen den Boss zum Ausgangspunkt des Gitters kommt wird es 3lebens des Spielers entnehmen.

## Türme

Die Türme werden auf ein Teil des Gitters gestellt. Die bewegen sich nicht und schießen auf der Creeps die vorbei gehen. Die verschiedenen Türme haben verschiedenen Schussgeschwindigkeit und DPS und Range.

### Schussgeschwindigkeit

Eigenschaft für den Turm. Wie schnell die den Schussen nach einander starten.

### DPS (Damage per Second)

Wieviel Schaden macht den Turm, jede Sekunde. Den ganzen an zahl wir von den Zielcreep Leben genommen.

### Upgrades

Noch eine Eigenschaft die für die Türme implementier werde. Wen der Spieler Klick auf den Turm, soll ein Popup kommen welcher fragt, ob der Turm verbessert wird. Die Upgrades kosten ingame-currency.

## Waves

Die Creeps kommen in Rudel. Es hängt ab von den Wavenummer, wie stark die Gegner werden. Die Rudel bestehen aus verschiedene Anzahl vom Creeps.

## User Interface

Es wird ein Menü enthalten, die durch „Еsc“ Knopfdruck aktiviert wird. Im Spiel in der obere Rechte Ecke steht die Information wieviel Geld momentan der Spieler hat wieviel Lebens(Live)

### Points

Soll zeigen wieviel ingame-currency der Spieler im Moment hat. Der kann die benutzen um neue Turme zu kaufen oder schon existierende zu verbessern.

### Live

Live zeigt wieviel Leben der Spieler noch hat. Wen des Lebens auf 0 fallen soll das Spiel beendet werden.

## Sound

Den Sound wird am Anfang des Spiels geladen und auf will gespielt.

## Animation

Das sind den tweens die benutzt werden um den Animation zu machen, Auf eine Entity wird den path ausgerechnet und danach für jede Grid teil wird ein neues tween gemacht um eine fleißigere Bewegung zu bekommen. Wen ein neues Tower gestellt wird, jede bewegliche Entity berechnet für sich selbst neu den Pfad zum Ziel.

# Praktische Lösung

## CreateJS

Für die Lösung den oben geschriebene Probleme wird die Suite von CreateJS benutzt. Die sind verantwortlich für die Entstehung von interaktiven Content in open web Technologien via HTML.

CreateJS ist von gskinner entwickelt und offiziell gesponsert bei Adobe, Microsoft, Mozilla und AOL.

In den CreateJS Suite gibt’s vier Haupt Bibliotheken.

### EaselJS

Stellt Entwicklung für Arbeit mit Grafiken und Interaktivität über HTML5 Canvas. Es stellt eine API zur Verfügung, die ganz nah zu den Flash ist. Die API unterstützt eine hierarchische Schau List und DOM Level 2 Events.

### TweenJS

Ist eine tweenende Bibliothek die mit JavaScript benutzt kann. Es ist entwickelt zu funktionieren mit den EaselJS Bibliothek, aber es ist nicht abhängig davon. Was Interessantes dabei ist, das Tweens kann implementiert sein für Nummer Objekten Eigenschaften sowie CSS style Eigenschaften.

### SoundJS

Es ist verantwortlich für den Audio Playback via HTML5, Web Audio oder Flash. Die Bibliothek schaut der Hardware auf der es geladen ist und lade ein passendes Plug-In für die Audio Playback. Die Bibliothek ist cross-platform das bedeutet es kann geladen sein auf meistens die moderne Browsers.

### PreloadJS

Ist der Bibliothek die vorladet verschiedene Dateien für schnellere und problemlose Anwendung. Es kann leicht Bilder, Töne, Text Dateien in das Projekt importieren. Es erlaub mehrfach Zugriff und mehrfach Verbindungen.

## Entwicklung

In den folgenden Teil des Berichts wird vorgestellt was die drei Studenten entwickelt haben. Die GIT Seite des Projekts ist: <http://github.com/groupTD/TD>

### Deian Dimitrov

Der Student hat sich beschäftigt mit der Entstehung den Spielgitter, die Bewegung von Creeps auf den Gitter, Debugging und Refactoring. Er hat sich auch mit diesen Bericht beschäftigt.

**Tile Entstehung:**

**function** Tile(grid, x, y, tileNumber, arrayX, arrayY) {  
 **this**.grid = grid;  
 **this**.x = x;  
 **this**.y = y;  
 **this**.blocked = 1;  
 **this**.number = tileNumber;  
 **this**.arrayX = arrayX;  
 **this**.arrayY = arrayY;  
}

**Grid Entstehung:**

Grid.prototype.init = **function** () {  
 **this**.tiles = **new** Array(**this**.horTilesCount);  
  
 **for** (**var** i = 0; i < **this**.horTilesCount; i++) {  
 **this**.tiles[i] = **new** Array(**this**.verTilesCount);  
 }  
 **var** tileNumber = 1;  
 **for** (**var** n = 0, y = **this**.gridYStartCoord; n < **this**.verTilesCount; n++, y += **this**.verTilesLength) {  
 **for** (**var** i = 0, x = **this**.gridXStartCoord; i < **this**.horTilesCount; i++, x += **this**.horTilesLength) {  
 //Setting grid frame blocked for enemies  
 **if**(i==0||n==0||i==**this**.horTilesCount-1||n==**this**.verTilesCount-1){  
 **if**(i==0&&n==Math.*floor*((**this**.verTilesCount)/2)){  
 **this**.tiles[i][n] = **new** Tile(**this**, x, y, tileNumber, i, n, 1);  
 }  
 **else if**(i==**this**.horTilesCount-1&&n==6)**this**.tiles[i][n] = **new** Tile(**this**, x, y, tileNumber, i, n, 1);  
 **else this**.tiles[i][n] = **new** Tile(**this**, x, y, tileNumber, i, n, 0);  
 }  
 **else this**.tiles[i][n] = **new** Tile(**this**, x, y, tileNumber, i, n, 1);  
 tileNumber++;  
 }  
 }  
}

**Grid Draw function:**

Grid.prototype.draw = **function** (stage) {  
 **function** drawLine(x, y, grid) {  
 **var** line = **new** createjs.Shape();  
 **if** (y === 0)line.graphics.*moveTo*(x, grid.gridYStartCoord).setStrokeStyle(1).beginStroke("#000000").lineTo(x, grid.verTilesLength \* grid.verTilesCount + grid.gridYStartCoord);  
 **else** line.graphics.*moveTo*(grid.gridXStartCoord, y).setStrokeStyle(1).beginStroke("#000000").lineTo(grid.horTilesLength \* grid.horTilesCount + grid.gridXStartCoord, y);  
 stage.addChild(line);  
 }  
  
 **for** (**var** x = **this**.gridXStartCoord; x < width; x += **this**.horTilesLength) {  
 drawLine(x, 0, **this**);  
 }  
 **for** (**var** y = **this**.gridYStartCoord; y < height; y += **this**.verTilesLength) {  
 drawLine(0, y, **this**);  
 }  
  
 **function** drawCircle(tile) {  
 **var** circle = **new** createjs.Shape();  
 circle.graphics.setStrokeStyle(1).beginStroke("rgba(255,0,0,1)").drawCircle(tile.x, tile.y, 2);  
 stage.addChild(circle);  
 }  
  
 **function** writeTileNumber(tile) {  
 **var** txt = **new** createjs.Text(tile.number, "15px Arial", "#000");  
 txt.x = tile.x + (0.25 \* tile.grid.horTilesLength);  
 txt.y = tile.y + (0.25 \* tile.grid.verTilesLength);  
 stage.addChild(txt);  
 }  
  
  
 **for** (**var** n = 0; n < **this**.verTilesCount; n++) {  
 **for** (**var** i = 0; i < **this**.horTilesCount; i++) {  
 drawCircle(**this**.tiles[i][n]);  
 writeTileNumber(**this**.tiles[i][n]);  
 }  
 }  
  
  
}

Funktion zum Berechnung den kürzesten Weg. Dabei wird die Bibliothek javascript-astar 0.4.0 benutzt. Folgende Informationen über den Bibliothek man kann finden unter: <https://github.com/bgrins/javascript-astar>

**Shortest Path Methode:**

Entity.prototype.getShortestPath = **function** (grid, startPoint, endPoint) {  
 **var** graph = **new** Array(grid.horTilesCount);  
  
 **for** (**var** i = 0; i < grid.horTilesCount; i++) {  
 graph[i] = **new** Array(grid.verTilesCount);  
 }  
  
 **for** (**var** x = 0; x < grid.horTilesCount; x++) {  
 **for** (**var** y = 0; y < grid.verTilesCount; y++) {  
 graph[x][y] = grid.tiles[x][y].blocked;  
 }  
 }  
 **var** startTile = **this**.getTile(grid, startPoint.x, startPoint.y);  
 **var** endTile = **this**.getTile(grid, endPoint.x, endPoint.y);  
 **var** graph2 = **new** Graph(graph);  
 **var** start = graph2.grid[startTile.arrayX][startTile.arrayY];  
 **var** end = graph2.grid[endTile.arrayX][endTile.arrayY];  
 **var** result = astar.*search*(graph2, start, end);  
  
 **return** result;  
}

Bekomme den Tile Objekt von gegeben Point(x, y), wird auch realisiert für schnellere binden Entity Objekt zu den Gitter.

**GetTile Funktion:**

Entity.prototype.getTile = **function** (grid, xSearch, ySearch) {  
  
 **function** Point(x, y) {  
 **this**.x = x;  
 **this**.y = y;  
 }  
  
 **function** Rectangle(x, y, width, height) {  
 **this**.x = x;  
 **this**.y = y;  
 **this**.width = width;  
 **this**.height = height;  
 }  
  
 **function** rectangleContainsPoint(rect, point) {  
 **if** (rect.*width* <= 0 || rect.*height* <= 0) {  
 **return false**;  
 }  
 **return** (point.x >= rect.x && point.x < rect.x + rect.*width* && point.y >= rect.y && point.y < rect.y + rect.*height*);  
 }  
  
 **for** (**var** xi = 0; xi < grid.horTilesCount; xi++) {  
 **for** (**var** yi = 0; yi < grid.verTilesCount; yi++) {  
 **var** point = **new** Point(xSearch, ySearch);  
 **var** rectangle = **new** Rectangle(grid.tiles[xi][yi].x, grid.tiles[xi][yi].y, grid.verTilesLength, grid.horTilesLength);  
 **if** (rectangleContainsPoint(rectangle, point)) {  
 **return** grid.tiles[xi][yi];  
 }  
 }  
 }  
}

### Valeri Vatchev

Der Student hat sich beschäftigt mit der Entstehung den Türme und ihren Projectiles

**Tower Entstehung:**

**function** Tower(game, params){  
 Entity.call(**this**, game, params);  
 **this**.tween = [];  
 **this**.kills = 0;  
 **this**.projSpeed = 0;  
}

**Tower Initialisierung:**

Tower.prototype.init = **function** (stage) {  
 // call base method  
 Entity.prototype.init.call(**this**, stage);  
 //make tower  
 **this**.initTower();  
};

**Tower Default Werte:**

Tower.prototype.initTower = **function** (stage){  
 **this**.angle = 0;  
 **this**.health = 10;  
 **this**.target = **null**;  
 **this**.id = Tower.prototype.idGen++;  
 **this**.cooldown = 4;  
 **this**.kills = 0;  
 **this**.damage = 0;  
 **this**.projSpeed = 100;  
  
};

**Türme schießen Funktion:**

Tower.prototype.shoot = **function**(){  
 **var** speed = Tower.prototype.projSpeed;  
 **var** math = [Math.cos(**this**.angle), Math.sin(**this**.angle), -Math.sin(**this**.angle), Math.cos(**this**.angle)];  
 **for**(**var** i = -1; i <= 1; i += 2){  
 **var** offset = mattvp(math, [0, i \* 5]);  
 **var** towerProj = **new** Projectile(**this**.game, **this**.x + offset[0], **this**.y + offset[1], speed \* math[0], speed \* math[1], **this**.angle, **this**);  
 towerProj.damage = Math.pow(1.2, **this**.level);  
 **this**.game.addProjectile(towerProj);  
 }

**this**.cooldown = **this**.getCooldownTime();  
};

**Projectile Entstehung:**

**function** Projectile(game,x,y,vx,vy,angle,tower){  
 **this**.game = game;  
 **this**.x = x;  
 **this**.y = y;  
 **this**.vx = vx;  
 **this**.vy = vy;  
 **this**.angle = angle;  
 **this**.tower = tower;  
 **this**.team = tower.team;  
 **this**.damage = 1;  
 **this**.vanished = **false**;  
}

**Tower Entfernen:**

Tower.prototype.dispose = **function** (stage) {  
 Entity.prototype.dispose.call(**this**, stage);  
  
 createjs.Tween.removeTweens(**this**.bitmap);  
};

Diese Turmstruktur ermöglicht eine einfache Erweiterung der Arten und der Fähigkeiten der Türme. Dies ermöglicht die Zugabe von anderen Typen for Türme mit verschiedenen Wirkungen.

### Martin Gebov

Der Student hat sich beschäftigt mit der Entstehung den Creeps, die Menu und die CreepWellen. Auch mit Debugging und Refactoring. Er hat sich auch mit diesen Bericht beschäftigt.

**Enemy constructor**

function Enemy(game, params) {

Entity.call(this, game, params);

this.speed = params.speed;

this.tween = [];

this.path;

this.health = params.health;

this.damage = params.damage;

}

**Pause feindliche Bewegung**

Enemy.prototype.pauseMovement = function() {

for(var i=0; i<this.tween.length;i++){

this.tween[i].setPaused(true);

}

//createjs.Tween.removeTweens(this.bitmap);

}

**Fortsetzen feindliche Bewegung**

Enemy.prototype.resumeMovement = function() {

for(var i=0; i<this.tween.length;i++){

this.tween[i].setPaused(false);

} //this.tween.setPaused(false);

}

**Fügen Sie einen Feind ins Spiel ein**

Game.prototype.addEnemy = function () {

if (this.enemies.length < this.enemyCount) {

var enemy = new Enemy(this, {

texturePath: "assets/enemy.png",

speed: 1000,

x: 0,

y: 0

});

this.enemies.push(enemy);

enemy.init(this.stage);

//this.enemies.push(enemy);

return enemy;

}

return null;

};

**Entfernen Sie einen Feind aus dem Spiel**

Game.prototype.removeEnemy = function (enemy) {

var enemyIndex = 0;

for (var i = 0; i < this.enemies.length; i++) {

if (this.enemies[i] === enemy) {

enemyIndex = i;

break;

}

}

// remove element from array

this.enemies.splice(enemyIndex, 1);

enemy.dispose(this.stage);

};

**Signalisieren Sie das Spiel, daß der Feind sein Ziel erreicht hat**

Game.prototype.enemyFinished = function (enemy) {

this.removeEnemy(enemy);

};

**Pause das Spiel, um das Menü zu zeigen**

Game.prototype.pause = function() {

this.paused = true;

for (var i = 0; i < this.enemies.length; i++) {

this.enemies[i].pauseMovement();

}

}

**Das Spiel fortsetzen**

Game.prototype.resume = function() {

this.paused = false;

for (var i = 0; i < this.enemies.length; i++) {

this.enemies[i].resumeMovement();

}

}

**Legen Sie einige Tasten das Spiel zu pausieren**

function keyBoardHandler(event) {

// when 'p' is pressed (for pause)

if (event.keyCode == 80) {

if (!game.paused) {

game.pause();

stage.addChild(getPauseMenu());

stage.update();

}

else {

stage.removeChild(getPauseMenu());

game.resume()

}

} else if (event.keyCode == 27) { // on escape

game.dispose();

stage.removeAllChildren();

initMenu();

}

}

function getPauseMenu() {

if (pauseMenu == null) {

pauseMenu = new createjs.Container();

var popupGraphics = new createjs.Graphics();

popupGraphics.beginFill('black').drawRect(width / 2 - 150, height / 2 - 150, 300, 300);

var popupShape = new createjs.Shape(popupGraphics);

pauseMenu.addChild(popupShape);

var pauseText = createText("Game paused");

pauseText.x = width / 2 - 130;

pauseText.y = height / 2 - 30;

pauseMenu.addChild(pauseText);

}

return pauseMenu;

}

# Zusammenfassung

Während der Entstehung, haben die Entwickler sich mit neue Technologien kennengelernt. JavaScript war eine neue Programmiersprache für der Studenten. Es hat viel mitgebracht für die Weitere professionelle Bildung des Studenten als Software Ingenieurs.

Mit die Entwicklung, entsteht ein Maze Tower Defence Spiel. Die Creeps(Gegner) laufen Ein Gitter, mit Hilfe von A\* Algorithmus für Berechnung den kürzesten Weg zum Ziel. Während die Gegner laufen, die Türme die von Spieler am Gitter gestellt sind, schießen die. Wen die Gegner den Ziel erreichen wird von den Lives des Spielers entnommen. Das Spiel wird gewonnen wen die letzte Wave auf Es gibt noch Funktionen die geschrieben werden kann. Die werden nötig sein, wen den Studenten, diesen Projekt als ein Vorsprung für den Master Thesis gemacht haben.

Das Spiel lauft in ein Web Browser aber es kann implementiert werden in Android oder IOS Umgebungen. Da sollen weitere Bibliotheken benutzt werden, um es als Einzige Mobile Anwendung für diese Betriebssystemen funktionierend zu machen.

# Literaturverzeichnis

[1] CreateJS Bibliothek; URL: <https://en.wikipedia.org/wiki/CreateJS>; Abgerufen: 28.6.2015

[2] CreateJS Official Seite; URL: <http://www.createjs.com/>; Abgerufen 28.6.2015

[3] Tower Defence Spiele; URL: <https://en.wikipedia.org/wiki/Tower_Defense>; Abgerufen 28.6.2015

[4] Kongregate Game Portal; URL: <http://www.kongregate.com/tower-defense-games> Abgerufen 28.6.2015

[5] CreateJS Community Page; URL: <http://community.createjs.com/>; Abgerufen: 28.6.2015

[6] Deutsch Portal für Tower Defence Spiele: URL:http://www.tower-defense-spiele.de/entstehung-und-entwicklung-von-tower-defense-spielen/ Abgerufen: 28.6.2015