**Project arbeit Mobile computing ss21**

**Defend the castle**

Kais Boutouri 535766

Mahmoud bouzgarou 535557

**Einführung:**

Wir haben ein einfaches Spiel auf der Android-Plattform mit der SurfaceView-Klasse nur durch verwalteten Java-Code erstellt. Das Spiel heißt BAD WIZARD, das ist ein 2D-Schießspiel.Sie sind ein Zauberer mit einem erstaunlichen Zauberkraft und sie können Ihres Power nutzen um zu überleben und die Zombie-Angriffe zu stoppen.

Fahren Sie Ihren Weg durch eine Zombie-Apokalypse.

Jeder wird dieses magische Schießspiel gegen Zombies mit 2D Charakteren und großartigen Effekten lieben und genießen und Höhe Punktzahl zu erreichen.

Dieses spiel hat eine Display-Interaktion durch Touch-Ereignisse (On Touch Event).

das Spiel siehst so aus:



**Steuerung:**

-Zauberer bewegen: Berühren Sie den halben Bildschirm auf der linken Seite, um zu bewegen

-Aktion: Schießen:: Berühren Sie den halben Bildschirm auf der rechten Seite , um auf Feinde zu schießen.



**Game Over (Spiel Vorbei):**

Sie verlieren das Spiel, wenn einer der Feinde Ihnen berührt oder er Ihren Bereich erreicht.

**PLAY THIS GAME AND BEAT THE HIGH SCORE**

**Anwendungen:**

**Full screen Mode: (VollbildModus)**

Wir haben den immersiven Modus verwendet, um den Vollbildmodus bei der Hauptaktivität (main activity) zu aktivieren damit wir die Nutzer im Vollbildmodus tiefer in die Inhalte einbinden und schützen Sie vor dem versehentlichen Verlassen der App.

Um den immersiven Modus zu aktivieren, haben wir SETSYSTEM\_UI\_VISIBILITY () aufgerufen in Verbindung mit Flag SYSTEM\_UI\_FLAG\_FULLSCREEN und SYSTEM\_UI\_FLAG\_HIDE\_NAVIAGTION .

**On Touch Event:**

Um die Spielsteuerung durch Berühren des Touchscreens einzustellen haben wir die ´´On Touch Event ´´ Methode angewendet.

Die Basisklasse für die Unterstützung von Berührungen ist die Klasse MotionEvent, die über die Methode On Touch Event( ) übergeben. Die enthält die ScreenX-Koordinate als einzige berührungsbezogenen Information.

if (event.getX() < *screenX* / 2) {   
*wizzard*.isGooingUp= true ;}

if (event.getX() > *screenX* / 2 )  
 *wizzard*.toAttack ++ ;

**Hauptfunktionen:**

Der Speilcode ist durch 3 Hauptteile unterteilt:

Update:

Die Methode Update() aktualisiert alle Objekte unseres Spiels. Sie bringt die Zombies voran aber sie aktualisiert jedoch nicht das Wizard-Objekt, da sich dessen Verschiebungswert nur ändert, wenn der Benutzer den Bildschirm berührt.

Draw:

Die Methode Draw (Convas Convas) wird nach Aufruf von Update () aufgerufen. Diese Methode zeichnet alle Objekte, die für die Spielanzeige relevant sind, auf die angegebene Leinwand.

Draw () zeichnet die Concas von Background 1 und 2, The Wizard,The Score (Spielstand),Zombies und Spells(der Zauber).

Sleep:

Schließlich wird die Leinwand entsperrt und für eine bestimmte Zeit in dem Ruhestand versetzt,de FPS (Frames per Sekunde) genannt wird.

Unser Gehirn verarbeitet etwa 30 bis 60 Bilder per Sekunde, Unseres Spiel aktualisiert dem Bildschirm alle 17 Millisekunde also 60 FPS

Nach der Adressierung der 3 Hauptfunktionen, erstellen wir 3 verschiedene Klassen für jedes bewegliche Objekt im Spiel, den Hintergrund, Feinde oder genannt Zombies, die Hauptfigur, die Zaubersprüche .

Es gibt **4 Gemeinsamkeiten** zwischen alle diesen Klassen,

1. Varibalen :

Jede Klasse hat 2 Variablen, um die Position des Objekts nach der x-und y-Achsen , Breite und Höhe und eine Variable des Typs Bitmap zu definieren. mit der Methode bitmapfactory.decode Ressourcen(), um ein Bild zu jedem Objekt anhängen

Z.B :

zombie1 = BitmapFactory.*decodeResource*(res ,R.drawable.*zombie1*);

1. Hit Box (getHitbox function) :

Jedes Objekt hat ein Rechteck um sich herum, um mit der Funktion gethitbox() unter Verwendung der Parameter

(x, y, x+Breite und y + Höhe) als Hitbox fungiert,

Rect getHitBox() {

return new Rect(x , y , x + width , y + height ) ;

}

iii. Ausgangsposition :

Jedes Objekt hat eine bestimmte Ausgangsposition auf dem Bildschirm, indem man die x- und y-Position des Objekts ändert Zum Beispiel können wir die anfängliche Position des Zauberers auf der y-Achse auf screenY und x auf screenX-**100 Pixel** setzen, um ihn auf die linke Seite des Bildschirms zu bringen.

y = screenY; //starting position in y aches\_ ;

x = (int) (*screenRatioX* - 100); // starting position in x aches

die Anfangsposition von Feinden ist etwas kompliziert denn die Y achse muss randamisiert werden und wird in der update() Funktion angegeben

zombie.x = *screenX* ; // place the Zombies at the right side of the screen  
zombie.y =random.nextInt(*screenY*-zombie.height) ; //randomize the y-achse of zombies

Die Illusion der Animation wird durch eine Schleife von 2 oder mehr if-Bedingungen erzeugt, wobei jede Schleife eine **Bitmap** zurückgibt mit hilf ein **counter** ,

Bitmap getZombie () {  
 if (zombieCounter == 1 ) {  
 zombieCounter ++ ;  
 return zombie1 ;  
 }  
 if (zombieCounter == 2 ) {  
 zombieCounter ++ ;  
 return zombie2 ;  
 }  
 if (zombieCounter == 3 ) {  
 zombieCounter ++ ;  
 return zombie3 ;  
 }  
  
 zombieCounter = 1 ;  
  
 return zombie4 ;

Die update() funktion in der GameView-Klasse des Spiels ist dafür verantwortlich, die Position des Objekts in jeder Schleife zu ändern :

1. Background :

Wird die Position des Hintergrunds 1 und 2 auf x-achse jedes Mal verringert, wenn die update()\_methode aufgerufen wird , und es wird eine if-Schleife erstellt, um das Hintergrundbild erneut anzeigen wenn es nicht auf dem Bildschirm ist

background1.x -= 10 \* *screenRatioX*;

background2.x -= 10 \* *screenRatioX*;

1. Wizzard :

Mit Hilfe der isGoingup (Boolean) , die auf **true** gesetzt wird, wenn der Benutzer auf die rechte Seite des Bildschirms klickt, können wir die Position des Wizzards auf der y-Achse um **+- 30 Pixel** verändern, um die Sprunganimation zu erstellen,

1. Spells

Die Zaubersprüche werden als **Liste** des Typs spell verwendet, jeder Zauber wird in diese Mülleimerliste verschoben, wenn er den Bildschirm verlässt.

for (Spell spell : spells ) {  
 if(spell.x > *screenX*)  
 trash.add(spell);  
 spell.x += 50 \**screenRatioX*;

Zombies

die Feinde werden als **array** verwendet, jeder Feind hat eine Geschwindigkeit, die mit random

type randomisiert, die Geschwindigkeit wird zwischen

3 \**screenRatioX* und 2 \* *screenRatioX* gesetzt

zombie.speed = random.nextInt(bound) ;

if (zombie.speed > ( 3 \* *screenRatioX*) ) // minimum speed of zombies  
 zombie.speed = (int) (2 \* *screenRatioX*); //set the zombies speed to 2 if its higher than 3  
if (zombie.speed < 1) //set the zombies speed to 2 if its lower than 1  
 zombie.speed = (int) (2 \* *screenRatioX*);

Mit Hilfe der getHitbox() Funktion jedes Objekts und der intersect Methode in der rect Klasse

können wir die **Game Over** Bedingung definieren, wenn die Rechtecke von **Wizzard** und

**Zombies** sich überschneiden isGameover ist auf **true** gesetzt

if (Rect.*intersects*(*wizzard*.getHitBox() , zombie.getHitBox())) {  
 isGameOver = true;;  
 return;

wenn sich die 2 Hitboxen von Zombies und Speels überschneiden, wird die Position der

Zombies aus dem Bildschirm verschoben und und bewertet den Spieler mit einem Punkt neu **(score)**

f (Rect.*intersects*( zombie.getHitBox() , spell.getHitBox())) {  
 *score*++ ; //reword the player with 1 point  
 zombie.x = -500 ; //dead zombies will stay out of the screen  
 spell.x =*screenX* +500 ; //used spells will stay out if the screen  
 zombie.wasShot = true ;  
}

Der letzte Teil ist die draw()funktion, die hauptsächlich die Methode drawBitmap() in der Klasse cavass verwendet,

canvas.drawBitmap(background1.background, background1.x,background1.y,paint);

Zombies und Zaubersprüche sind Arrays, also wird mit einer einfachen for-Schleife gezeichnet

for (Zombie zombie :zombies) {  
 canvas.drawBitmap(zombie.getZombie() , zombie.x , zombie.y , paint);  
}

wenn das Spiel zu Ende ist, wird der boolsche Wert isplaying auf **false** gesetzt,

und 2 Funktionen saveifhighscore() und waitbeforeexisting() werden aufgerufen

und das Bild des toten Zauberers wird gezeichnet

if (isGameOver) {  
  
 isplaying = false;  
 canvas.drawBitmap(*wizzard*.getDead(), *wizzard*.x, *wizzard*.y, paint);  
 getHolder().unlockCanvasAndPost(canvas);  
 saveIfHighScore() ;  
 waitBeforeExiting();  
 return;