Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Уфимский государственный нефтяной технический университет

Кафедра вычислительной техники и инженерной кибернетики

КУРСОВАЯ РАБОТА

по дисциплине

Мобильные приложения и программирование устройств

ТЕМА ПРОЕКТА

2D - платформер

Выполнил студент группы БПОи-18-01 Э.И. Гумеров

Принял ст. преподаватель Е.В. Дружинская

Дата представления работы: \_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Результат: \_\_\_\_\_\_\_\_\_\_

Уфа, 2021 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc41649562)

[1. Обзор предметной области 4](#_Toc41649563)

[2. Проектирование приложения 5](#_Toc41649564)

[2.1 Функциональное моделирование 5](#_Toc41649565)

[2.2 Эскизирование 5](#_Toc41649566)

[2.3 Логическое моделирование 6](#_Toc41649567)

[3. Программная реализация 7](#_Toc41649568)

[ЗАКЛЮЧЕНИЕ 16](#_Toc41649569)

# ВВЕДЕНИЕ

Мобильные приложения заняли огромный рынок в сфере мобильных приложений, для того, чтобы привлечь конечного пользователя, разработчик должен создать что-то новое или старое, но лучше, поэтому было решено разработать игру с понятным интерфейсом и открытым исходным кодом и максимально упростить добавление новых элементов в нее даже для тех, кто не захочет разбираться во всем игровом коде.

# Обзор предметной области

В Google Play есть огромное количество приложений на любую тематику, в том числе игры, особенно 2D платформеры, так как их легко писать и они как правило пишутся на специальных для этого средах разработки, которые в том или ином случае урезают возможности.

Если писать на “чистом” языке, то можно реализовать почти все на что хватит фантазии, поэтому планируется создать игру с открытым и удобным исходным кодом, в которую можно будет добавлять все, на что способен язык, в данном случае java.

# Проектирование приложения

## 2.1 Функциональное моделирование

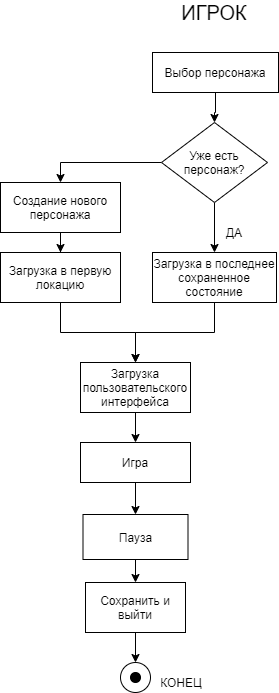


Рисунок 1 – Диаграмма использования приложения

2.2 Эскиз экрана



Рисунок 2 – Эскиз экрана

* 1. Диаграмма классов

На данной диаграмме представлены основные классы, необходимые для работы приложения.

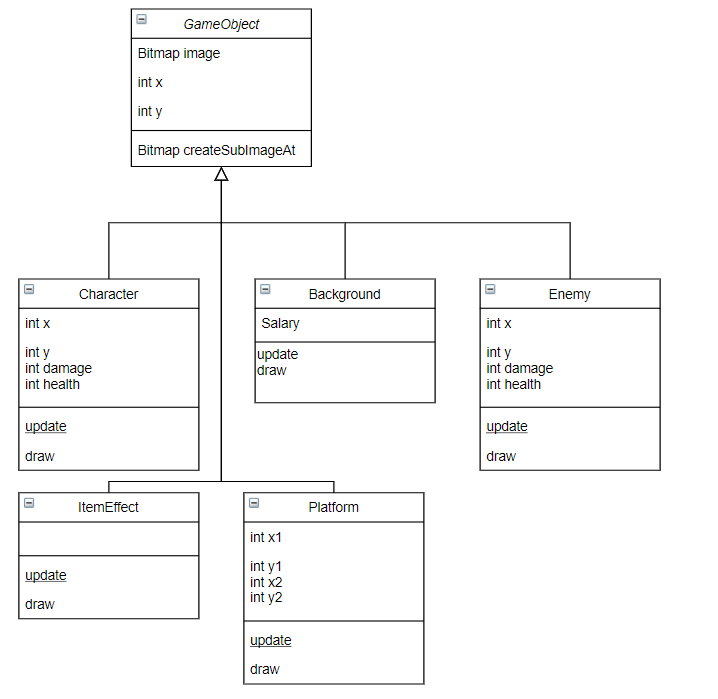


Рисунок 3 – диаграмма классов.

1. Программная реализация

В качестве кода предоставим классы GameSurface.java и Character.java

public class GameSurface extends SurfaceView implements SurfaceHolder.Callback {  
  
 public GameThread gameThread;  
  
 private final List<Character> characterList = new ArrayList<Character>();  
 private final List<Explosion> explosionList = new ArrayList<Explosion>();  
 private final List<BackGround> bgList=new ArrayList<BackGround>();  
 private final List<MoveButton> moveButtonList=new ArrayList<MoveButton>();  
 public final List<Enemy> enemyList=new ArrayList<Enemy>();  
 public final List<Platform> platforms=new ArrayList<Platform>();  
 private static final int *MAX\_STREAMS*=100;  
 private int soundIdExplosion;  
 private int soundIdBackground;  
 private Rect characterButtons;  
 private int fingerOnCharButtons;  
 private boolean hasStarted=false;  
 public Level curLevel;  
 public String curLevelInfo=null;  
  
  
 private boolean soundPoolLoaded;  
 public SoundPool soundPool;  
 public float gravity=0.001f;  
  
  
 public GameSurface(Context context) {  
 super(context);  
  
 // Make Game Surface focusable so it can handle events.  
 this.setFocusable(true);  
  
 // Sét callback.  
 this.getHolder().addCallback(this);  
  
 this.initSoundPool();  
 }  
  
 private void initSoundPool() {  
 // With Android API >= 21.  
 if (Build.VERSION.*SDK\_INT* >= 21 ) {  
  
 AudioAttributes audioAttrib = new AudioAttributes.Builder()  
 .setUsage(AudioAttributes.*USAGE\_GAME*)  
 .setContentType(AudioAttributes.*CONTENT\_TYPE\_SONIFICATION*)  
 .build();  
  
 SoundPool.Builder builder= new SoundPool.Builder();  
 builder.setAudioAttributes(audioAttrib).setMaxStreams(*MAX\_STREAMS*);  
  
 this.soundPool = builder.build();  
 }  
 // With Android API < 21  
 else {  
 // SoundPool(int maxStreams, int streamType, int srcQuality)  
 this.soundPool = new SoundPool(*MAX\_STREAMS*, AudioManager.*STREAM\_MUSIC*, 0);  
 }  
  
 // When SoundPool load complete.  
 this.soundPool.setOnLoadCompleteListener(new SoundPool.OnLoadCompleteListener() {  
 @Override  
 public void onLoadComplete(SoundPool soundPool, int sampleId, int status) {  
 soundPoolLoaded = true;  
  
 // Playing background sound.  
 playSoundBackground();  
 }  
 });  
  
 // Load the sound background.mp3 into SoundPool  
 this.soundIdBackground= this.soundPool.load(this.getContext(), R.raw.*ponos*,1);  
  
 // Load the sound explosion.wav into SoundPool  
 this.soundIdExplosion = this.soundPool.load(this.getContext(), R.raw.*explosion*,1);  
  
  
 }  
  
 public void playSoundExplosion() {  
 if(this.soundPoolLoaded) {  
 float leftVolumn = 0.8f;  
 float rightVolumn = 0.8f;  
 // Play sound explosion.wav  
 int streamId = this.soundPool.play(this.soundIdExplosion,leftVolumn, rightVolumn, 1, 0, 1f);  
 }  
 }  
  
 public void playSoundBackground() {  
 if(this.soundPoolLoaded) {  
 float leftVolumn = 0.8f;  
 float rightVolumn = 0.8f;  
 // Play sound background.mp3  
 int streamId = this.soundPool.play(this.soundIdBackground,leftVolumn, rightVolumn, 1, -1, 1f);  
 }  
 }  
  
 @Override  
 public boolean onTouchEvent(MotionEvent event) {  
 for(Character c:characterList){  
 return c.checkUI(event);  
 }  
 return false;  
 }  
  
  
  
  
  
  
  
  
  
 public void update() {  
 for(BackGround backGround: this.bgList) {  
 backGround.update();  
 }  
 for(Character character: characterList) {  
 character.update();  
 }  
 for(Explosion explosion: this.explosionList) {  
 explosion.update();  
 }  
  
 for(MoveButton button: this.moveButtonList){  
 button.update();  
 }  
  
 for(Enemy enemy:this.enemyList){  
 enemy.update();  
 }  
  
  
 Iterator<Explosion> iterator= this.explosionList.iterator();  
 while(iterator.hasNext()) {  
 Explosion explosion = iterator.next();  
  
 if(explosion.isFinish()) {  
 // If explosion finish, Remove the current element from the iterator & list.  
 iterator.remove();  
 continue;  
 }  
 }  
  
 }  
  
 @Override  
 public void draw(Canvas canvas) {  
 super.draw(canvas);  
  
 for(BackGround backGround: this.bgList){  
 backGround.draw(canvas);  
 }  
 for(Platform platform:this.platforms){  
 platform.draw(canvas);  
 }  
 for(Character chibi: characterList) {  
 chibi.draw(canvas);  
 }  
  
 for(Explosion explosion: this.explosionList) {  
 explosion.draw(canvas);  
 }  
  
 for(Enemy enemy:this.enemyList){  
 enemy.draw(canvas);  
 }  
  
 }  
  
   
   
 // Implements method of SurfaceHolder.Callback  
 @Override  
 public void surfaceCreated(SurfaceHolder holder) {  
 if(!hasStarted) {  
 curLevel=new Lev\_1(this);  
 curLevel.init();  
  
 Bitmap chibiBitmap1 = BitmapFactory.*decodeResource*(this.getResources(), R.drawable.*chibi1*);  
 Character chibi1 = new Class\_SwordMan(this, chibiBitmap1, 100, 50);  
  
// Bitmap chibiBitmap2 = BitmapFactory.decodeResource(this.getResources(), R.drawable.chibi2);  
// Character chibi2 = new Character(this, chibiBitmap2, 300, 150);  
  
 Bitmap background = BitmapFactory.*decodeResource*(this.getResources(), R.drawable.*cave\_bg*);  
 BackGround bc = new BackGround(this, background, this.getWidth(), this.getHeight());  
  
  
  
  
  
 this.bgList.add(bc);  
 this.characterList.add(chibi1);  
// this.characterList.add(chibi2);  
  
  
 hasStarted=true;  
 }  
  
 this.soundPool.autoResume();  
 this.gameThread = new GameThread(this,holder);  
 this.gameThread.setRunning(true);  
 this.gameThread.start();  
 }  
  
 // Implements method of SurfaceHolder.Callback  
 @Override  
 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {  
  
 }  
  
 // Implements method of SurfaceHolder.Callback  
 @Override  
 public void surfaceDestroyed(SurfaceHolder holder) {  
 boolean retry= true;  
 {  
 try {  
 this.soundPool.autoPause();  
 this.gameThread.setRunning(false);  
  
 // Parent thread must wait until the end of GameThread.  
 this.gameThread.join();  
  
 }catch(InterruptedException e) {  
 e.printStackTrace();  
 }  
 retry=true;  
 }  
 }  
  
 public void createPlatform(Bitmap image,int x1,int y1,int x2,int y2){  
 platforms.add(new Platform(image,x1,y1,x2,y2));  
 }  
 public void clearLevel(){  
 this.enemyList.clear();  
 this.platforms.clear();  
 }  
  
 public List<Enemy> checkForTargets(Rect rect){  
 List<Enemy> currentTargets=new ArrayList<Enemy>();  
 for(Enemy enemy:enemyList){  
 if(rect.contains(enemy.stickyX,enemy.stickyY)||(enemy.getRect().contains(rect.left,rect.top)))currentTargets.add(enemy);  
 }  
 return currentTargets;  
 }  
}

public abstract class Character extends GameObject {  
  
 private static final int *ROW\_TOP\_TO\_BOTTOM* = 0;  
 private static final int *ROW\_RIGHT\_TO\_LEFT* = 1;  
 private static final int *ROW\_LEFT\_TO\_RIGHT* = 2;  
 private static final int *ROW\_BOTTOM\_TO\_TOP* = 3;  
 protected final List<ItemEffect> effectList=new ArrayList<ItemEffect>();  
 // Row index of Image are being used.  
 private int rowUsing = *ROW\_LEFT\_TO\_RIGHT*;  
 private boolean isOnSurface=true;  
 private int colUsing;  
 private int jumpsLeft=1;  
 private int stickyX;  
 private int stickyY;  
 private int distx;  
 private int yHalf;  
 public int baseAttackDamage; public int addedAttackDamage;  
 public int baseStrength; public int addedStrength;  
 public int baseAgility; public int addedAgility;  
 public int baseIntelligence; public int addedIntelligence;  
 public int baseEndurance; public int addedEndurance;  
 public int baseAttackSpeed;  
 public boolean attack=true;  
 public List<Buff> buffs=new ArrayList<Buff>();  
  
 private Bitmap[] leftToRights;  
 private Bitmap[] rightToLefts;  
 private Bitmap[] topToBottoms;  
 private Bitmap[] bottomToTops;  
  
 // Velocity of game character (pixel/millisecond)  
 public static final float *VELOCITY* = 1.2f;  
  
 private int movingVectorX = 0;  
 private int movingVectorY = 0;  
 private float gravity;  
 private float VELOCITYY=0.18f;  
 private long lastDrawNanoTime =-1;  
 protected int facing=1;  
 private Location curLoc;  
  
 protected GameSurface gameSurface;  
 public int lastStickyX;  
 public int lastStickyY;  
  
 public Character(GameSurface gameSurface, Bitmap image, int x, int y) {  
 super(image, 4, 3, x, y);  
 this.gameSurface= gameSurface;  
 this.curLoc=gameSurface.curLevel.currentLoc;  
 this.gravity=gameSurface.gravity;  
 this.topToBottoms = new Bitmap[colCount]; // 3  
 this.rightToLefts = new Bitmap[colCount]; // 3  
 this.leftToRights = new Bitmap[colCount]; // 3  
 this.bottomToTops = new Bitmap[colCount]; // 3  
  
 for(int col = 0; col< this.colCount; col++ ) {  
 this.topToBottoms[col] = this.createSubImageAt(*ROW\_TOP\_TO\_BOTTOM*, col);  
 this.rightToLefts[col] = this.createSubImageAt(*ROW\_RIGHT\_TO\_LEFT*, col);  
 this.leftToRights[col] = this.createSubImageAt(*ROW\_LEFT\_TO\_RIGHT*, col);  
 this.bottomToTops[col] = this.createSubImageAt(*ROW\_BOTTOM\_TO\_TOP*, col);  
 }  
 }  
  
 public Bitmap[] getMoveBitmaps() {  
 switch (rowUsing) {  
 case *ROW\_BOTTOM\_TO\_TOP*:  
 return this.bottomToTops;  
 case *ROW\_LEFT\_TO\_RIGHT*:  
 return this.leftToRights;  
 case *ROW\_RIGHT\_TO\_LEFT*:  
 return this.rightToLefts;  
 case *ROW\_TOP\_TO\_BOTTOM*:  
 return this.topToBottoms;  
 default:  
 return null;  
 }  
 }  
  
 public Bitmap getCurrentMoveBitmap() {  
 Bitmap[] bitmaps = this.getMoveBitmaps();  
 return bitmaps[this.colUsing];  
 }  
  
  
 public void update() {  
 facing=movingVectorX>0?1:movingVectorX<0?-1:facing;  
 Iterator<ItemEffect> iterator=effectList.iterator();  
 attack();  
 while(iterator.hasNext()) {  
 ItemEffect effect = iterator.next();  
 effect.update();  
 if(effect.isFinish()) {  
 iterator.remove();  
 continue;  
 }  
 }  
 this.colUsing++;  
 if(colUsing >= this.colCount) {  
 this.colUsing =0;  
 }  
 // Current time in nanoseconds  
 long now = System.*nanoTime*();  
  
 // Never once did draw.  
 if(lastDrawNanoTime==-1) {  
 lastDrawNanoTime= now;  
 }  
 // Change nanoseconds to milliseconds (1 nanosecond = 1000000 milliseconds).  
 int deltaTime = (int) ((now - lastDrawNanoTime)/ 1000000 );  
  
 // Distance moves  
 float distance = *VELOCITY* \* deltaTime;  
  
 double movingVectorLength = Math.*sqrt*(movingVectorX\* movingVectorX);  
 if(!isOnSurface) movingVectorY += VELOCITYY\*deltaTime+gravity\*(deltaTime\*deltaTime);  
 else {movingVectorY=0; jumpsLeft=1;}  
 switch (isOutOfSurface()){  
 case -1:  
 break;  
 case 0:  
 x=lastStickyX;  
 y=lastStickyY;  
 break;  
 case 1:  
 if(curLoc.Left!=null)curLoc=curLoc.Left;  
 curLoc.start();  
 x=gameSurface.getWidth();  
 break;  
 case 2:  
 if(curLoc.Right!=null)curLoc=curLoc.Right;  
 curLoc.start();  
 x=x-gameSurface.getWidth();  
 break;  
 case 3:  
 if(gameSurface.curLevel.currentLoc.Down!=null)(gameSurface.curLevel.currentLoc=gameSurface.curLevel.currentLoc.Down).start();  
 y=0;  
 break;  
 case 4:  
 if(gameSurface.curLevel.currentLoc.Up!=null)(gameSurface.curLevel.currentLoc=gameSurface.curLevel.currentLoc.Up).start();  
 y=gameSurface.getHeight();  
 break;  
 }  
 // Calculate the new position of the game character.  
 distx= (int)(distance\* movingVectorX / movingVectorLength);  
 yHalf=y+getHeight()/2;  
 int lastY = y;  
 this.x = x+distx;  
 this.y = y + Math.*min*(movingVectorY,40);  
 this.stickyX=x+getWidth()/2;  
  
 this.stickyY=y+getHeight();  
 isOnSurface=false;  
 for(Platform platform:gameSurface.platforms){  
  
 if (platform.rect.contains(stickyX, stickyY)&&lastY<=platform.y1) {  
 isOnSurface = true;  
 y = platform.y1 - getHeight();  
 lastStickyX = x;  
 lastStickyY = y;  
 }  
  
  
 if(platform.rect.contains(stickyX,stickyY-3)&&!platform.rect.contains(stickyX-distx,stickyY-3)){  
 this.x=x-distx;  
 }  
 }  
  
  
 if( movingVectorX > 0 ) {  
 if(movingVectorY > 0 && Math.*abs*(movingVectorX) < Math.*abs*(movingVectorY)) {  
 this.rowUsing = *ROW\_TOP\_TO\_BOTTOM*;  
 }else if(movingVectorY < 0 && Math.*abs*(movingVectorX) < Math.*abs*(movingVectorY)) {  
 this.rowUsing = *ROW\_BOTTOM\_TO\_TOP*;  
 }else {  
 this.rowUsing = *ROW\_LEFT\_TO\_RIGHT*;  
 }  
 } else if(movingVectorX<0){  
 if(movingVectorY > 0 && Math.*abs*(movingVectorX) < Math.*abs*(movingVectorY)) {  
 this.rowUsing = *ROW\_TOP\_TO\_BOTTOM*;  
 }else if(movingVectorY < 0 && Math.*abs*(movingVectorX) < Math.*abs*(movingVectorY)) {  
 this.rowUsing = *ROW\_BOTTOM\_TO\_TOP*;  
 }else {  
 this.rowUsing = *ROW\_RIGHT\_TO\_LEFT*;  
 }  
 }  
  
 }  
  
 public void draw(Canvas canvas) {  
 Bitmap bitmap = this.getCurrentMoveBitmap();  
 canvas.drawBitmap(bitmap,x, y, null);  
 UI(canvas);  
 for(ItemEffect effect:effectList){  
 effect.draw(canvas);  
 }  
  
 // Last draw time.  
 this.lastDrawNanoTime= System.*nanoTime*();  
 }  
  
 public void setMovingVector(int movingVectorX, int movingVectorY) {  
 this.movingVectorX= movingVectorX;  
 this.movingVectorY = movingVectorY;  
 }  
 public void jump(){  
 if(jumpsLeft>0) {  
 jumpsLeft--;  
 isOnSurface = false;  
 setMovingVectorY(-50);  
 }  
 }  
  
 public void setMovingVectorX(int movingVectorX) {  
 this.movingVectorX=movingVectorX;  
 }  
  
 public void setMovingVectorY(int movingVectorY) {  
 this.movingVectorY=movingVectorY;  
 }  
 public void setOnSurface(boolean state){  
 isOnSurface=state;  
 }  
  
 public abstract void Ability1();  
  
 public abstract void Ability2();  
  
 public abstract void UI(Canvas canvas);  
  
 public abstract void evaluateStats();  
  
 public abstract void attack();  
  
 public abstract boolean checkUI(MotionEvent event);  
  
 public int isOutOfSurface(){  
 if(x<0-getWidth())return 1;  
 if(x>gameSurface.getWidth())return 2;  
 if(y>gameSurface.getHeight())return 0;  
 if(y<0-getHeight())return 4;  
 return -1;  
 }  
 public abstract void evaluateBuffs();  
 public void annul(){  
 addedAttackDamage=0;  
 addedStrength=0;  
 addedAgility=0;  
 addedIntelligence=0;  
 addedEndurance=0;  
 }  
}

ЗАКЛЮЧЕНИЕ

Была создана начальная версия игры, в которой уже доступны примитивы перемещения, механика нанесения урона, особые способности персонажа и переход по локациям. В дальнейшем планируется добавлять в нее новые механики и постепенная оптимизация кода.