DESIGN PROJECT

Flutter ile Mobil Oyun Geliştirme

Arş.Gör.Dr. FATMA ÖZGE ÖZKÖK

HALİL İBRAHİM MÖHÜRLÜ 1030516701
YUSUF SEYİTOĞLU 1030516739

İçindekiler

DESİGN PROJECT FİNAL RAPORU
1 GİRİŞ

Erciyes Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü bünyesinde yer alan Design Project dersi kapsamında gerçekleştirdiğimiz proje 2 kişilik proje grubumuz tarafından eşzamanlı olarak yürütüldü.

Amaç

Raporda geliştirilen mobil tank oyununun yazılımının tasarım, gerçekleştirme ve sonuç süreçleri detayları, projenin amacı

Raporda geliştirilen mobil tank oyunu hakkında projenin amacı, yazılımı, gereksinimleri, yazılımın çalışacağı ortam, nasıl bir ara yüze sahip olacağı tasarım, gerçekleştirme, sonuç olarak 3 bölümde incelenecektir.

Projede Kullanılan Yazılım Donanım Araçları

Uygulamayı geliştirirken ihtiyacımız olan bazı yazılımlara ihtiyaç duyduk.

-Visual studio code

Visual studio code, projenin gerçekleştirildiği araçtır.

-Dart dili

Dart dili, açık kaynaklı ve genel amaçlı olan programlama dili olup web, sunucu, mobil uygulamalar ve IoT geliştirilebilir.

-Flutter sdk

Flutter, mobil uygulama geliştirmek için kullanılan bir kullanıcı arayüzü geliştirme aracıdır. Android, ios, Windows, Mac ve Linux için uygulama geliştirmede kullanılır.

-Bonfire

Flutterda mobil oyun geliştirmek için kullanılan flame kütüphanesinin üzerine geliştirilmiş rpg(role play gaming) tarzında piksel art oyun geliştirme kütüphanesidir

-Tiled

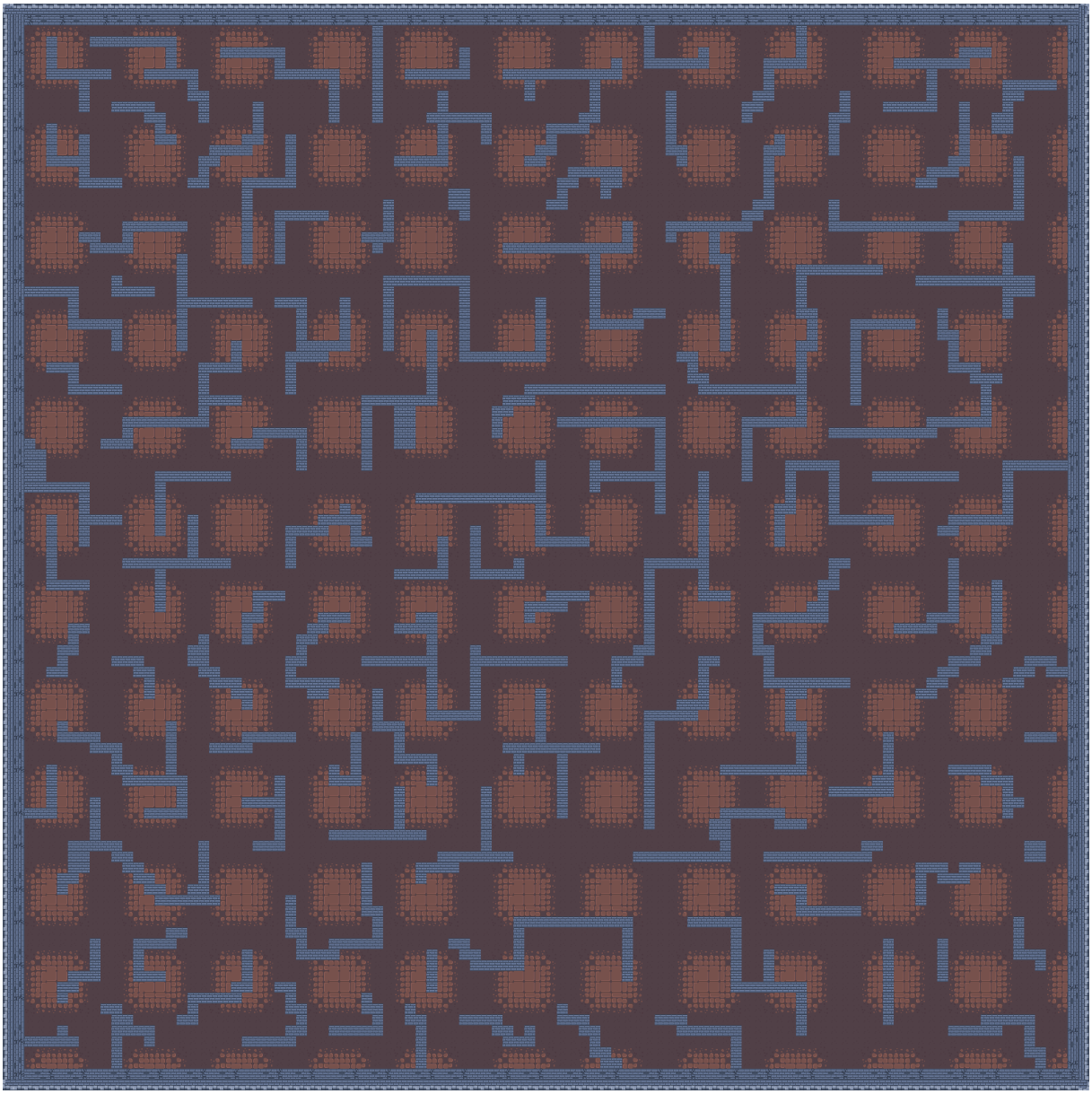
2 boyutlu oyunlar için piksel şeklinde haritalar oluşturmak ve detaylandırmak için geliştirilen bir yazılımdır.

2 GELİŞME

Kodların gerçekleştirilmesi

Map

Oyunun geçtiği harita Tiled programı kullanılarak tasarlandı. 100x100 birim kareden oluşan harita ile oyuncunun hareket edebileceği mesafenin arttırılması amaçlandı. Ayrıca haritaya duvarlar da eklenerek labirent benzeri bir görünüm oluşturuldu böylece oyuncu daha fazla alanda hareket ederek taktikler uygulayabilir ve hayatta kalma şansını arttırabilir. Aşağıda tasarlanmış haritanın fotoğrafı görünmektedir.



Enemy

Bu sınıfta oyunda oyuncuya saldıracak olan düşman karakterleri kodlandı. Düşmanların hasarı ve can değerleri belirlendi. Düşman sınıfı SimpleEnemy sınıfndan kalıntı alınarak gerçekleştirildi ve ayrıca ObjectCollision sınıfının metotları da implemente edildi. Yapıcı metotla düşman karakterlerin uzunluğu ve genişliği ve hareket hızı ayarlandı. setupCollision metodu ile düşman karakterlerin oyuncu ile hangi alan içerisinde etkileşime gireceği belirlendi.

```
class EnemyTank extends SimpleEnemy with ObjectCollision {
  final Vector2 initPosition;
  double attack = 25;

  EnemyTank(this.initPosition)
      : super(
          animation: EnemySpriteSheet.tankAnimations(),
          position: initPosition,
          width: tileSize * 0.8,
          height: tileSize * 0.8,
          speed: tileSize / 0.35,
          life: 120,
        ) {
    setupCollision(
      CollisionConfig(
        collisions: [
          CollisionArea.rectangle(
            size: Size(
              valueByTileSize(7),
              valueByTileSize(7),
            ),
            align: Vector2(valueByTileSize(3), valueByTileSize(4)),
          ),
        ],
      ),
    );
  }
```

```dart
@override
void render(Canvas canvas) {
  this.drawDefaultLifeBar(
    canvas,
    borderRadius: BorderRadius.circular(2),
  );
  super.render(canvas);
}

@override
void update(double dt) {
  super.update(dt);

  this.seeAndMoveToPlayer(
    closePlayer: (player) {
      execAttack();
    },
    radiusVision: tileSize * 4,
  );
}

@override
void die() {
  gameRef.add(
    AnimatedObjectOnce(
      animation: GameSpriteSheet.smokeExplosion(),
      position: this.position,
    ),
  );
  removeFromParent();
  super.die();
}
```

SimpleEnemy sınıfının metotlarını implemente ederek düşman karakterlere bazı özellikler kazandırdık. Bu özellikler oyuncuyu gördüğünde oyuncuya yönelmesi, saldırması ve canı tükendiğinde ölmesi yani oyundan çıkarılmasıdır. Bu özellikler SimpleEnemy sınıfının metotlarını override ederek gerçekleştirildi. Bu metotlardan bazıları aşağıdaki gibidir.

render metodu ile geçilen canvas objesine düşman karakterlerini çizdirdik.

update metodu ile düşman karakterin her framede güncellenmesini sağladık ve seeAndMoveToPlayer metodu ile düşman karakterlerin oyuncu gördüğünde execAttack metodu ile saldırmasını sağladık.

die metodu ile düşman karakterin öldüğünde bulunduğu konumda patlama efekti oluşturmasını ve oyundan çıkarılmasını sağladık.

```
  void execAttack() {
    this.simpleAttackMelee(
      height: tileSize * 0.62,
      width: tileSize * 0.62,
      damage: attack,
      interval: 800,
      attackEffectBottomAnim: EnemySpriteSheet.enemyAttackEffectBottom(),
      attackEffectLeftAnim: EnemySpriteSheet.enemyAttackEffectLeft(),
      attackEffectRightAnim: EnemySpriteSheet.enemyAttackEffectRight(),
      attackEffectTopAnim: EnemySpriteSheet.enemyAttackEffectTop(),
      execute: () {
        Sounds.attackEnemyMelee();
      },
    );
  }

  @override
  void receiveDamage(double damage, dynamic id) {
    this.showDamage(
      damage,
      config: TextPaintConfig(
        fontSize: valueByTileSize(5),
        color: Colors.white,
        fontFamily: 'Normal',
      ),
    );
    super.receiveDamage(damage, id);
  }
}
```

Bu metotlar ile düşman karakterlerin saldırı yaparken işletilecek efektler ve saldırının ne kadar bir alana etki edeceğini belirledik. receiveDamage metodu ile düşman karakterlerin hasar almasını sağladık ebeveyn sınıfın receiveDamage metoduna geçirilen id parametresi ile idsi verilen düşman karakterin hasar almasını sağladık.

Player

Bu sınıfta oyuncuyu temsil edecek karakterin özellikleri kodlandı. SimplerPlayer sınıfı kalıtıldı, Lighting ve ObjectCollision sınıflarıyla karaktere ışıklandırma ve çevre ile etkileşime girme özellikleri kazandırıldı.

```
class PlayerTank extends SimplePlayer with Lighting, ObjectCollision {
  final Vector2 initPosition;
  double attack = 25;
  double stamina = 100;
  double initSpeed = tileSize / 0.25;
  async.Timer _timerStamina;
  bool showObserveEnemy = false;

  PlayerTank({this.initPosition,}) : super(
          animation: PlayerSpriteSheet.playerAnimations(),
          width: tileSize,
          height: tileSize,
          position: initPosition,
          life: 200,
          speed: tileSize / 0.25,
        ) {
    setupCollision(
      CollisionConfig(
        collisions: [
          CollisionArea.rectangle(
            size: Size(valueByTileSize(8), valueByTileSize(8)),
            align: Vector2(
              valueByTileSize(4),
              valueByTileSize(8),
            ),
          ),
        ],
      ),
    );

  setupLighting(
    LightingConfig(
      radius: width * 1.5,
      blurBorder: width,
      color: Colors.deepOrangeAccent.withOpacity(0.2),
    ),
  );
}
```

setupLighting metodu ile karaktere genişliğinin 1.5 katı genişlikte ışıklandırma eklendi ve renk değeri Colors sınıfından 0.2 opaklık değeri ile alındı.

```dart
@override
void joystickChangeDirectional(JoystickDirectionalEvent event) {
  this.speed = initSpeed * event.intensity;
  super.joystickChangeDirectional(event);
}

@override
void joystickAction(JoystickActionEvent event) {
  if (event.id == 0 && event.event == ActionEvent.DOWN) {
    actionAttack();
  }

  if (event.id == 1 && event.event == ActionEvent.DOWN) {
    actionAttackRange();
  }
  super.joystickAction(event);
}
```

joystick metotlarını override ederek karakterin joystick ile hareket etmesi sağlandı. Karakterin hareket hızı ise event.intensity değerine göre ayarlandı böylece joystick ne kadar fazla hareket ettirilirse karakter o kadar hızlı hareket etmektedir.

```dart
@override
void die() {
  removeFromParent();
  gameRef.addGameComponent(
    GameDecoration.withSprite(
      Sprite.load('player/crypt.png'),
      position: Vector2(
        this.position.center.dx,
        this.position.center.dy,
      ),
      height: 30,
      width: 30,
    ),
  );
  super.die();
}
```

die metodu ile karakterin oyundan çıkarılması ve öldüğü yerde mezar taşının çıkması sağlandı. Mezar taşı oyuna yeni bir GameDecoration componenti olarak eklendi.

```
void actionAttackRange() {
  if (stamina < 10) {
    return;
  }
  decrementStamina(10);
  this.simpleAttackRange(
    animationRight: GameSpriteSheet.fireBallAttackRight(),
    animationLeft: GameSpriteSheet.fireBallAttackLeft(),
    animationUp: GameSpriteSheet.fireBallAttackTop(),
    animationDown: GameSpriteSheet.fireBallAttackBottom(),
    animationDestroy: GameSpriteSheet.fireBallExplosion(),
    width: tileSize * 0.65,
    height: tileSize * 0.65,
    damage: 10,
    speed: initSpeed * (tileSize / 32),
    enableDiagonal: false,
    destroy: () {
      Sounds.explosion();
    },
    collision: CollisionConfig(
      collisions: [
        CollisionArea.rectangle(size: Size(tileSize / 2, tileSize / 2)),
      ],
    ),
    lightingConfig: LightingConfig(
      radius: tileSize * 0.9,
      blurBorder: tileSize / 2,
      color: Colors.deepOrangeAccent.withOpacity(0.4),
    ),
  );
}
```

Bu metot ile karakterin saldırı yapması sağlandı. Karakterin stamina değeri 10'dan az ise saldırı yapması bsit bir if bloğu ile engellendi eğer saldırı yapabiliyor ise stamina değeri decrementStamina metodu ile 10 birim azaltıldı. Yapılan saldırının alanı, verdiği hasar, saldırı hızı, çarpışma alanı ve ışıklandırılması gerçekleştirildi.

```dart
@override
void update(double dt) {
  if (isDead) return;
  _verifyStamina();
  this.seeEnemy(
    radiusVision: tileSize * 6,
    notObserved: () {
      showObserveEnemy = false;
    },
    observed: (enemies) {
      if (showObserveEnemy) return;
      showObserveEnemy = true;
      _showEmote();
    },
  );
  super.update(dt);
}
```

Update metodu ile karakterin her frame de güncellenmesi, düşmanları görmesi için bir görüş alanı belirtildi ve eğer öldüyse güncellenmesinin durdurulması sağlandı.

verifyStamina metodu ile karakterin stamina değerinin timer sınıfı ile her 150ms de 2 birim arttırılması sağlandı. Ayrıca stamina değeri kontrol edilerek eğer 100 birimi aşıyorsa tekrar 100 birime eşitlenmesi gerçekleştirildi.

```dart
void _verifyStamina() {
  if (_timerStamina == null) {
    _timerStamina = async.Timer(Duration(milliseconds: 150), () {
      _timerStamina = null;
    });
  } else {
    return;
  }

  stamina += 2;
  if (stamina > 100) {
    stamina = 100;
  }
}
```

```dart
void decrementStamina(int i) {
  stamina -= i;
  if (stamina < 0) {
    stamina = 0;
  }
}
```

Bu metot karakterin stamina değerini verilen parametre kadar azaltan eğer 0'dan küçük olursa tekrar 0 değerine eşitleyen yardımcı metottur.

```
@override
void receiveDamage(double damage, dynamic id) {
  if (isDead) return;
  this.showDamage(
    damage,
    config: TextPaintConfig(
      fontSize: valueByTileSize(5),
      color: Colors.orange,
      fontFamily: 'Normal',
    ),
  );
  super.receiveDamage(damage, id);
}
```

Bu metot ile karakterin hasar alması gerçekleştirildi. Eğer karakter ölmüş ise hasar alması durdurulur. Ölmemiş ise parametre olarak verilen id'ye göre karakterin hasar alması sağlanır.

Interface

Bu sınıf ile karakterin canını ve stamina değerini gösteren arayüz InterfaceComponent sınıfından kalıtım alınarak kodlandı. Ara yüzün konumu, genişliği, sağlık barının uzunluğu ve stamina barının uzunluğu gibi değerler değişkenlere atandı ve yapıcı metot ile ara yüzü oluşturacak resim yüklendi.

```
class BarLifeComponent extends InterfaceComponent {
  double padding = 20;
  double widthBar = 90;
  double strokeWidth = 12;

  double maxLife = 0;
  double life = 0;
  double maxStamina = 100;
  double stamina = 0;

  BarLifeComponent()
      : super(
          id: 1,
          position: Vector2(20, 20),
          sprite: Sprite.load('health_ui.png'),
          width: 120,
          height: 40,
        );
```

```
  @override
  void update(double t) {
    if (this.gameRef.player != null) {
      life = this.gameRef.player.life;
      maxLife = this.gameRef.player.maxLife;
      if (this.gameRef.player is PlayerTank) {
        stamina = (this.gameRef.player as PlayerTank).stamina;
      }
    }
    super.update(t);
  }

  @override
  void render(Canvas c) {
    try {
      _drawLife(c);
      _drawStamina(c);
    } catch (e) {}
    super.render(c);
  }
```

Update metodu ile stamina ve sağlık değerleri her frame için karakterden alınıp ilgili değişkenlere atandı ve render metodu ile ekrana çizdirildi. drawLife ve drawStamine metotları ise sağlık ve stamina barlarını renklendiren ve çukub şeklinde çizidirilmesini sağlayan yardımcı metotlardır.

Oyunun Yapılandırılması

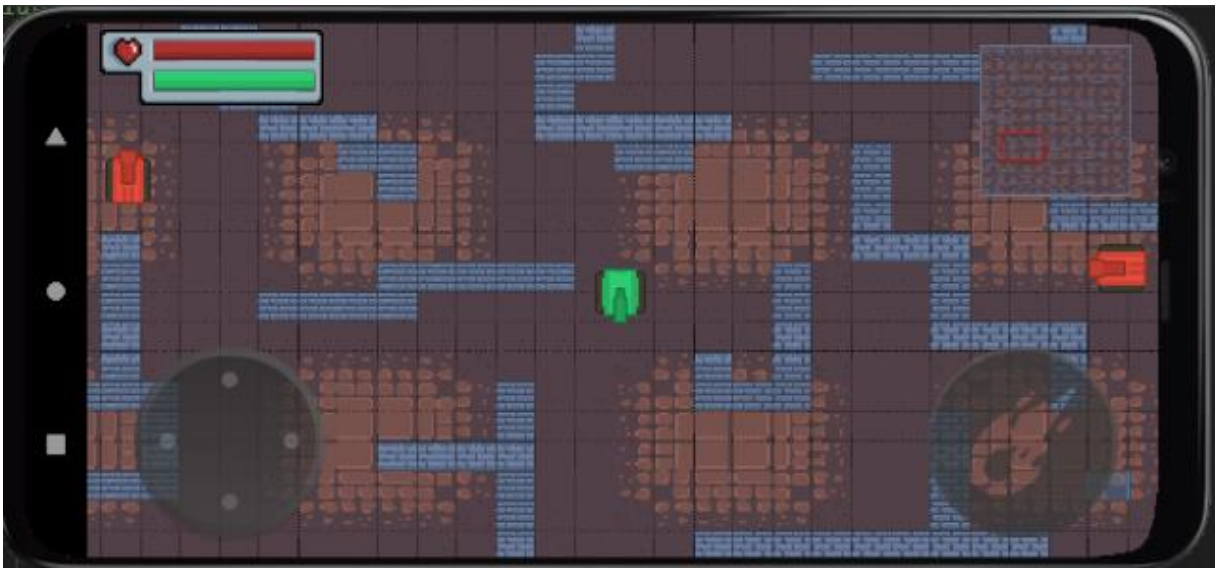
```
Widget build(BuildContext context) {
  Size sizeScreen = MediaQuery.of(context).size;
  tileSize = max(sizeScreen.height, sizeScreen.width) / 15;
  return Material(
    color: Colors.transparent,
    child: BonfireTiledWidget(
      gameController: _controller,
      joystick: Joystick(
        directional: JoystickDirectional(
          spriteBackgroundDirectional: Sprite.load('joystick_background.png'),
          spriteKnobDirectional: Sprite.load('joystick_knob.png'),
          size: 100,
          isFixed: false,
        ),
        actions: [
          JoystickAction(
            actionId: 0,
            sprite: Sprite.load('joystick_atack.png'),
            spritePressed: Sprite.load('joystick_atack_selected.png'),
            size: 80,
            margin: EdgeInsets.only(bottom: 50, right: 50),
          ),
          JoystickAction(
            actionId: 1,
            sprite: Sprite.load('joystick_atack_range.png'),
            spritePressed: Sprite.load('joystick_atack_range_selected.png'),
            size: 50,
            margin: EdgeInsets.only(bottom: 50, right: 160),
          )
        ],
      ),
      player: PlayerTank(
        initPosition: Vector2(2 * tileSize, 3 * tileSize),
      ),
      map: TiledWorldMap(
        'tiled/map.json',
        forceTileSize: Size(tileSize, tileSize)
      ),
      lightingColorGame: Colors.black.withOpacity(0.6),
      background: BackgroundColorGame(Colors.grey[900]),
    ),
  );
}
```

Build metodunun BonfireTiledWidget parametresine yukarıda kodlanmış sınıflar BonfireTiledWidget'ın uygun parametreleri ile eşleşecek şekilde geçirilerek oyun çalıştırılabilir hale gelir.

3 SONUÇ

Design Project dersi için uyguladığımız bu projede Flutter ile bir mobil uygulama geliştirdik. Uygulamayı geliştirirken ki amacımız Flutter'de deneyim sahibi olmak, mobil tabanlı uygulama geliştirmek ve kendimizi bazı konularda geliştirmekti.

Aldığımız Design Project dersi ile birçok konuda yeni terimler ve teknolojiler ile tanışma fırsatı bulduk. Bunları yaparken aynı zamanda Flutter ile yapılan uygulamaları analiz ettik, mobil uygulama geliştirme eğitim videolarını izleyip araştırmalar yaptık. Design Project dersi sayesinde takım olarak nasıl birlikte çalışılabilir, beraber nasıl uygulama geliştirilebileceğimizi deneyimledik.



Projede hedeflenen sonuç yukarıdaki görsel şeklindedir fakat bonfire kütüphanesindeki kaynak yetersizliğinden dolayı ve kütüphanenin güncel olmamasından dolayı hedeflediğimiz çıktıya ulaşamadık.

Kaynakça

[1] [Çevrimiçi]. Available: https://www.mapeditor.org/.

[2] [Çevrimiçi]. Available: https://code.visualstudio.com/.

[3] [Çevrimiçi]. Available: https://dart.dev/.

[4] [Çevrimiçi]. Available: https://flutter.dev/.

[5] [Çevrimiçi]. Available: https://github.com/RafaelBarbosatec/bonfire.