**README - פרויקט משחק פלטפורמה Desert Ball**

**פרטי הסטודנט**

**מרווא עבד אל רזאק :** 325604197  
**עמרו שויכי :** 212443485

**הסבר כללי :**

פרויקט זה מממש משחק פלטפורמה דו-ממדי בסביבת מדבר עם אלמנטים של פעולה והרפתקה. המשחק כולל שחקן שיכול לזוז, לקפץ ולירות, אויבים מסוגים שונים עם בינה מלאכותית מתקדמת, פריטים לאיסוף, מכשולים ואזורים חשוכים עם מערכת תאורה דינמית. המשחק נבנה על מנוע הפיזיקה Box2D ותומך במספר מסכים כולל תפריט, הגדרות, עזרה ומשחק. המשחק כולל מערכת הפתעות, מעברים בין שלבים, ואפקטים ויזואליים מתקדמים.

**תיכון (Design)**

**ארכיטקטורת המערכת**

המשחק בנוי על פי עקרונות OOP מתקדמים כולל:

* **Component-Entity System**: כל אובייקט במשחק הוא Entity המכיל רכיבים (Components) שמגדירים את התנהגותו
* **Strategy Pattern**: למערכת הבינה המלאכותית עם אסטרטגיות שונות
* **Observer Pattern**: לניהול אירועים ותגובות למשתמש
* **Command Pattern**: לניהול פעולות, ביטול פעולות והיסטוריה
* **Factory Pattern**: ליצירת אובייקטים ובניית ישויות
* **Singleton Pattern**: לניהול משאבים גלובליים וגישה למערכות
* **State Pattern**: לניהול מצבי שחקן שונים
* **Multimethod Pattern**: לטיפול בהתנגשויות בין סוגי ישויות

**חלוקת האחריות העיקרית**

**שכבת הליבה (Core Layer)**

* **Entity**: בסיס לכל אובייקט במשחק עם מערכת רכיבים
* **Component**: בסיס לכל רכיב עם lifecycle management
* **Transform**: ניהול מיקום, סיבוב וגודל
* **AppContext**: ניהול שירותים גלובליים עם Singleton
* **IScreen**: ממשק למסכים שונים
* **ScreenManager**: ניהול מעברים בין מסכים

**שכבת הרכיבים (Components Layer)**

* **RenderComponent**: אחראי על הצגה גרפית
* **PhysicsComponent**: אחראי על פיזיקה ואינטגרציה עם Box2D
* **HealthComponent**: אחראי על בריאות ונזק
* **MovementComponent**: אחראי על דפוסי תנועה שונים
* **AIComponent**: אחראי על בינה מלאכותית עם Strategy Pattern

**שכבת הישויות (Entities Layer)**

* **PlayerEntity**: ישות השחקן עם מערכות משנה מיוחדות
* **EnemyEntity**: בסיס לכל האויבים עם AI מתקדם
* **CollectibleEntity**: בסיס לפריטים לאיסוף
* **Environment Entities**: אובייקטי סביבה כמו קרקע, מכשולים, בארות
* **Combat Entities**: קליעים ומערכות לחימה

**שכבת המערכות (Systems Layer)**

* **EntityManager**: ניהול כל הישויות עם lifecycle management
* **PhysicsManager**: ניהול עולם הפיזיקה
* **CollisionManager**: זיהוי והתמודדות עם התנגשויות
* **RenderSystem**: מערכת הרינדור
* **EventSystem**: מערכת אירועים עם Observer Pattern
* **DarkLevelSystem**: מערכת תאורה ואפקטי חושך

**אינטראקציה בין האובייקטים**

1. **App** מתחיל את המשחק ויוצר את כל המנהלים
2. **GameSession** מתאמת בין כל המנהלים העיקריים
3. **EntityManager** מנהל את כל הישויות ומעדכן אותן
4. **PhysicsManager** מעדכן את עולם הפיזיקה
5. **CollisionManager** בודק התנגשויות ומפעיל handlers מתאימים
6. **EventSystem** מפיץ אירועים בין רכיבים שונים
7. **ScreenManager** מנהל מעברים בין מסכים שונים
8. **DarkLevelSystem** מנהל אפקטי תאורה ואפלה

**רשימת הקבצים שיצרנו**

**Application Layer**

* App.h/cpp - נקודת הכניסה הראשית לאפליקציה
* WindowManager.h/cpp - ניהול חלון SFML עם validation
* GameInitializer.h/cpp - אתחול מערכות המשחק
* GameLoop.h/cpp - לולאת המשחק הראשית עם frame timing
* AppCleanupManager.h/cpp - ניקוי משאבים בסיום
* AppContext.h/cpp - Service locator עם Singleton pattern

**Core Layer**

* Entity.h/cpp - מחלקת הבסיס לכל אובייקט במשחק
* Component.h/cpp - מחלקת הבסיס לכל רכיב
* Transform.h/cpp - רכיב עבור מיקום וסיבוב
* IScreen.h - ממשק למסכים שונים
* ScreenManager.h/cpp - ניהול מעברים בין מסכים
* ICommand.h - ממשק לפקודות עם Command Pattern
* CommandInvoker.h/cpp - מנהל פקודות עם Undo/Redo
* IButtonObserver.h - ממשק עבור Observer Pattern לכפתורים
* ButtonSubject.h/cpp - Subject עבור כפתורים
* SimpleObserver.h - גירסה פשוטה של Observer Pattern
* AudioSettingsManager.h/cpp - ניהול הגדרות שמע
* SettingsResourceManager.h/cpp - ניהול משאבים למסך הגדרות
* UITheme.h/cpp - ניהול נושאים ויזואליים
* ResourcePaths.h - קבועי נתיבים למשאבים
* Constants.h - קבועי המשחק הראשיים

**Components**

* RenderComponent.h/cpp - רכיב הצגה גרפית
* PhysicsComponent.h/cpp - רכיב פיזיקה עם Box2D
* HealthComponent.h/cpp - רכיב בריאות ונזק
* MovementComponent.h/cpp - רכיב תנועה עם דפוסים שונים
* AIComponent.h/cpp - רכיב בינה מלאכותית

**Entities - Player**

* PlayerEntity.h/cpp - ישות השחקן הראשית
* PlayerInputHandler.h/cpp - מעבד קלט השחקן
* PlayerScoreManager.h/cpp - מנהל ניקוד השחקן
* PlayerStateManager.h/cpp - מנהל מצבי השחקן
* PlayerVisualEffects.h/cpp - אפקטים ויזואליים של השחקן
* PlayerWeaponSystem.h/cpp - מערכת נשק השחקן

**Entities - Enemies**

* EnemyEntity.h/cpp - בסיס לכל האויבים
* SquareEnemyEntity.h/cpp - אויב מרובע שמתפצל
* FalconEnemyEntity.h/cpp - אויב מעופף עם אנימציה
* FalconWeaponSystem.h/cpp - מערכת נשק לאויב המעופף
* SmartEnemyEntity.h/cpp - אויב חכם עם AI מתקדם ועיניים

**Entities - Collectibles**

* CollectibleEntity.h/cpp - בסיס לפריטים לאיסוף
* CoinEntity.h/cpp - מטבעות עם תנועה מעגלית
* GiftEntity.h/cpp - מתנות עם סוגים שונים

**Entities - Environment**

* GroundEntity.h/cpp - רצפה וקרקע
* BoxEntity.h/cpp - קופסאות שניתן לדחוף
* CactusEntity.h/cpp - קקטוס שמזיק לשחקן
* FlagEntity.h/cpp - דגל סיום השלב
* SeaEntity.h/cpp - ים שהורג את השחקן
* WellEntity.h/cpp - באר למעבר לשלבים חשוכים

**Entities - Combat**

* ProjectileEntity.h/cpp - קליעים עם פיזיקה וכבידה

**States (State Pattern)**

* PlayerState.h/cpp - בסיס למצבי השחקן
* NormalState.h/cpp - מצב רגיל של השחקן
* BoostedState.h/cpp - מצב מואץ
* ShieldedState.h/cpp - מצב מוגן
* MagneticState.h/cpp - מצב מגנטי למשיכת מטבעות
* ReversedState.h/cpp - מצב הפוך
* HeadwindState.h/cpp - מצב רוח נגדית

**Systems - Events**

* EventSystem.h/cpp - מערכת אירועים עם Observer Pattern
* GameEvents.h/cpp - הגדרות אירועי המשחק
* GameEventCoordinator.h/cpp - מתאם אירועי המשחק

**Systems - Collision**

* MultiMethodCollisionSystem.h/cpp - מערכת התנגשויות עם Multimethod
* GameCollisionSetup.h/cpp - הגדרת handlers להתנגשויות

**Systems - Managers**

* EntityManager.h/cpp - ניהול כל הישויות
* PhysicsManager.h/cpp - ניהול עולם הפיזיקה
* CollisionManager.h/cpp - ניהול התנגשויות
* EntityCleanupManager.h/cpp - ניקוי ישויות לא פעילות
* LevelManager.h/cpp - ניהול רצף שלבים
* GameLevelManager.h/cpp - ניהול שלבים עם אירועים
* CameraManager.h/cpp - ניהול מצלמה עם מעקב אחרי שחקן
* SurpriseBoxManager.h/cpp - ניהול קופסאות הפתעה

**Systems - Rendering**

* RenderSystem.h/cpp - מערכת רינדור ראשית
* BackgroundRenderer.h/cpp - רינדור רקע
* DarkLevelSystem.h/cpp - מערכת תאורה ואפקטי חושך עם ray casting

**AI System**

* AIStrategy.h/cpp - אסטרטגיית AI בסיסית
* GuardStrategy.h/cpp - אסטרטגיית שמירה
* FollowStrategy.h/cpp - אסטרטגיית מעקב
* PatrolStrategy.h/cpp - אסטרטגיית סיור

**Commands**

* ChangeScreenCommand.h/cpp - פקודה למעבר בין מסכים
* EscapeKeyCommand.h/cpp - פקודה למקש Escape
* ExitGameCommand.h/cpp - פקודה ליציאה מהמשחק
* SettingsCommandHandler.h/cpp - מנהל פקודות למסך הגדרות

**Screens**

* MenuScreen.h/cpp - מסך התפריט הראשי
* GameplayScreen.h/cpp - מסך המשחק הראשי
* SettingsScreen.h/cpp - מסך הגדרות עם בקרת עוצמת קול
* LoadingScreen.h/cpp - מסך טעינה עם progress bar
* HelpScreen.h/cpp - מסך עזרה והוראות
* AboutScreen.h/cpp - מסך אודות המשחק

**UI Components**

* Button.h/cpp - כפתור בסיסי
* ObservableButton.h/cpp - כפתור עם Observer Pattern
* ButtonModel.h/cpp - מודל נתוני כפתור
* ButtonInteraction.h/cpp - טיפול באינטראקציות כפתור
* ButtonRenderer.h/cpp - רינדור כפתור
* ButtonFactory.h/cpp - יצירת כפתורים עם Factory Pattern
* Slider.h/cpp - מחוון גרירה
* VolumeControlPanel.h/cpp - פאנל בקרת עוצמת קול
* UIOverlay.h/cpp - שכבת UI במשחק
* UIObserver.h/cpp - Observer עבור אירועי UI
* SurpriseBoxScreen.h/cpp - מסך קופסת הפתעה עם אנימציות
* GameOverScreen.h/cpp - מסך סיום המשחק
* WinningScreen.h/cpp - מסך ניצחון
* MenuButtonManager.h/cpp - ניהול כפתורי התפריט
* MenuButtonObserver.h/cpp - Observer עבור כפתורי התפריט
* MenuAnimationController.h/cpp - בקר אנימציות התפריט
* MenuEventHandler.h/cpp - מטפל אירועי התפריט
* MenuRenderer.h/cpp - רינדור התפריט
* SettingsUIFactory.h/cpp - יצירת רכיבי UI להגדרות
* SettingsUIRenderer.h/cpp - רינדור UI להגדרות
* SettingsEventHandler.h/cpp - טיפול באירועי הגדרות

**Settings System**

* SettingsAutoSaveManager.h/cpp - ניהול שמירה אוטומטית
* SettingsCommandExecutor.h/cpp - מבצע פקודות בהגדרות
* SettingsEventLogger.h/cpp - רישום אירועי הגדרות
* SettingsInitializer.h/cpp - אתחול מערכת הגדרות
* SettingsInputHandler.h/cpp - טיפול בקלט בהגדרות

**Utilities**

* ResourceManager.h - ניהול משאבים (Template) עם exception handling
* LevelLoader.h/cpp - טעינת שלבים מקבצים
* InputService.h/cpp - שירות קלט עם מעקב אחרי מצב מקשים
* EntityFactory.h/cpp - יצירת ישויות עם Factory Pattern
* TileType.h - הגדרת סוגי אריחים

**Game Management**

* GameSession.h/cpp - מתאם המשחק הראשי

**Exceptions & Logging**

* Exception.h/cpp - מחלקת בסיס לחריגות עם source location
* GameExceptions.h/cpp - חריגות מיוחדות למשחק
* Logger.h/cpp - מערכת לוגים מתקדמת
* GameExceptions.h/cpp (Services) - חריגות פשוטות יותר
* Logger.h/cpp (Services) - לוגר פשוט

**Configuration**

* ScreenTypes.h - הגדרת סוגי מסכים
* CMakeLists.txt - קבצי בנייה למודולים שונים

**מבני נתונים עיקריים ותפקידיהם**

**EntityManager**

* std::unordered\_map<IdType, std::unique\_ptr<Entity>> m\_entities - מפה של כל הישויות במשחק
* שימוש בזיכרון יעיל ומאפשר גישה מהירה לישויות לפי ID
* Template-based creation עם perfect forwarding

**MultiMethodCollisionSystem**

* std::unordered\_map<CollisionKey, CollisionHandler> m\_handlers - מפה של handlers להתנגשויות
* מאפשר Multimethod Dispatch לטיפול בהתנגשויות בין סוגי ישויות שונים
* Hash-based lookup עם CollisionKeyHash לביצועים מיטביים

**EventSystem**

* std::unordered\_map<std::type\_index, ListenerList> m\_listeners - מפה של listeners לאירועים
* מאפשר הפצת אירועים יעילה למערכות שונות
* Type-safe subscription עם templates

**Component Storage**

* std::unordered\_map<std::type\_index, std::unique\_ptr<Component>> m\_components - רכיבים בכל Entity
* שימוש ב-type\_index לגישה מהירה לרכיבים לפי טיפוס
* RAII memory management

**Physics Integration**

* b2World - עולם הפיזיקה של Box2D
* כל PhysicsComponent מכיל b2Body\* המקושר לעולם הפיזיקה
* Automatic synchronization בין Transform ו-Physics

**Resource Management**

* std::unordered\_map<std::string, std::unique\_ptr<Resource>> m\_resources - משאבים עם lazy loading
* Exception-safe loading עם GameExceptions
* Template-based ResourceManager עבור סוגי משאבים שונים

**AI State Management**

* std::map<SmartStrategy, StrategyPerformance> m\_strategyHistory - היסטוריה של ביצועי AI
* Learning system עם success/failure tracking
* Strategy selection עם performance metrics

**Command Pattern Storage**

* std::vector<std::unique\_ptr<ICommand>> m\_commandHistory - היסטוריית פקודות
* Current index tracking עבור Undo/Redo
* Memory-limited history עם automatic cleanup

**אלגוריתמים הראויים לציון**

**Shadow Casting Algorithm (DarkLevelSystem)**

* **Raycasting**: יצירת 360 קרניים מנקודת האור בכל הכיוונים
* **Intersection Testing**: בדיקת חיתוך הקרניים עם מכשולים באמצעות line-segment intersection
* **Shadow Map Generation**: יצירת מפת צללים דינמית עם RenderTexture
* **Complexity**: O(n*m*r) כאשר n = מספר הקרניים, m = מספר המכשולים, r = מספר מקורות האור
* **Optimization**: Spatial partitioning עבור מכשולים גדולים

**AI Decision Making (SmartEnemyEntity)**

* **Threat Assessment**: הערכת רמת איום השחקן עם distance, health, ו-velocity factors
* **Terrain Analysis**: ניתוח יתרון השטח עם pathfinding primitives
* **Strategy Selection**: אלגוריתם בחירת אסטרטגיה אופטימלית עם machine learning
* **Learning System**: מערכת למידה עם success rate tracking וperformance metrics
* **Predictive Behavior**: חיזוי מיקום השחקן עם velocity extrapolation

**Collision Detection Optimization**

* **Broad Phase**: AABB-based spatial partitioning עם grid או quadtree
* **Narrow Phase**: בדיקה מדויקת של התנגשויות עם Box2D
* **Multimethod Dispatch**: O(1) lookup עבור collision handlers
* **Entity Filtering**: רק ישויות פעילות נכללות בבדיקה

**Multi-Method Dispatch Implementation**

* **Type-based Dispatch**: שימוש ב-std::type\_index עבור runtime type identification
* **Hash-based Lookup**: O(1) average case עם custom hash function
* **Symmetry Handling**: automatic handling של (A,B) ו-(B,A) orderings
* **Template Safety**: compile-time type checking עם static\_assert

**Resource Loading with Exception Safety**

* **Lazy Loading**: משאבים נטענים רק כשנדרשים
* **Exception Propagation**: structured exception handling עם context information
* **RAII Memory Management**: automatic cleanup עבור failed loads
* **Fallback Mechanisms**: graceful degradation עם default resources

**Player State Machine**

* **State Transitions**: finite state machine עם validation
* **Duration Tracking**: time-based state expiration
* **Effect Stacking**: multiple simultaneous effects management
* **Singleton States**: memory-efficient state instance management

**באגים ידועים**

**Known Issues**

1. **Physics Tunneling**: אובייקטים מהירים מאוד עלולים לחדור דרך קירות דקים
   * **Root Cause**: Discrete collision detection ב-Box2D
   * **Workaround**: הגדלת תדירות הפיזיקה או שימוש ב-Continuous Collision Detection
   * **Status**: תחת בדיקה עבור critical gameplay elements
2. **Shadow Flickering**: צללים עלולים לרפרף בתנאים מסוימים
   * **Root Cause**: דיוק נומרי בחישובי הקרניים וrounding errors
   * **Mitigation**: הגדלת epsilon values וstabilization filters
   * **Status**: משפר את החוויה ב-99% מהמקרים
3. **Memory Usage**: מערכת הצללים צורכת זיכרון רב במפות גדולות
   * **Root Cause**: multiple RenderTextures עבור shadow mapping
   * **Mitigation**: הגבלת מספר הקרניים, ray culling, ו-texture pooling
   * **Status**: optimized עבור רוב המכשירים
4. **AI Performance**: מספר רב של SmartEnemies עלול לגרום לירידה בביצועים
   * **Root Cause**: expensive decision-making calculations בכל frame
   * **Mitigation**: decision throttling, LOD system עבור AI complexity
   * **Status**: acceptable עד 10 smart enemies בו-זמנית

**Performance Considerations**

* מערכת הצללים דורשת GPU עם shader support טוב
* Large levels עם מאות entities עלולים לגרום לstuttering
* Memory fragmentation עלולה להתרחש אחרי משחק ממושך

**Platform-Specific Issues**

* **Windows**: font rendering עלולה להיות שונה על מערכות שונות
* **File Paths**: case sensitivity issues בסביבות Linux
* **Audio**: latency issues על מכשירים מסוימים

**הערות אחרות**

**הרחבות עתידיות**

1. **Networking Support**: תמיכה במשחק מרובה משתתפים עם client-server architecture
2. **Level Editor**: עורך שלבים גרפי עם real-time preview
3. **Advanced Shaders**: שיפור האפקטים הגרפיים עם custom GLSL shaders
4. **Sound System**: מערכת אודיו מתקדמת עם 3D positioning ו-environmental effects
5. **Mobile Support**: port למכשירים ניידים עם touch controls
6. **Steam Integration**: achievements, leaderboards, ו-cloud saves

**תלויות חיצוניות**

* **SFML 2.5+**: גרפיקה, אודיו, window management, ו-input handling
* **Box2D 2.4+**: מנוע פיזיקה עם collision detection ו-rigid body dynamics
* **C++20**: תכונות מתקדמות כמו concepts, modules, ו-source\_location
* **CMake 3.15+**: build system עם modern CMake practices

**Build Requirements**

* **Compiler**: GCC 10+ או Clang 12+ או MSVC 2019+
* **Platform**: Windows 10+, macOS 10.15+, או Linux (Ubuntu 20.04+)
* **RAM**: מינימום 4GB, מומלץ 8GB+
* **GPU**: OpenGL 3.3+ support עבור advanced rendering features

**הגדרות Build**

* **Debug Mode**: כולל assertion checking, extended logging, ו-memory leak detection
* **Release Mode**: full optimization עם -O3, strip symbols, ו-LTO
* **Profile Mode**: כלים לבדיקת ביצועים עם gprof או Visual Studio Profiler
* **Sanitizer Builds**: AddressSanitizer, ThreadSanitizer עבור bug detection

**משאבים נדרשים**

* **Textures**: כל הטקסטורות ב-resources/images/ (PNG format, powers of 2)
* **Fonts**: גופנים ב-resources/fonts/ (TTF format)
* **Levels**: קבצי שלבים ב-resources/levels/ (custom text format)
* **Audio**: קבצי שמע ב-resources/audio/ (OGG format מומלץ)
* **Shaders**: GLSL shaders ב-resources/shaders/ (אם משתמשים)

**Best Practices Used**

* **RAII**: Resource management עם smart pointers
* **Exception Safety**: Strong exception guarantees בכל המקומות הקריטיים
* **Const Correctness**: extensive use של const methods וparameters
* **Modern C++**: auto, range-based for loops, lambdas, ו-smart pointers
* **Single Responsibility**: כל class עם תפקיד ברור ומוגדר
* **Dependency Injection**: loose coupling בין components

**Testing Strategy**

* **Unit Tests**: עבור critical algorithms כמו collision detection
* **Integration Tests**: עבור inter-system communication
* **Performance Tests**: עבור AI algorithms ו-rendering systems
* **Memory Tests**: עבור leak detection ו-memory usage patterns

**Documentation**

* **Doxygen Comments**: comprehensive API documentation
* **Architecture Diagrams**: UML diagrams עבור major systems
* **Code Style**: consistent formatting עם clang-format
* **Git History**: meaningful commit messages עם conventional commits