IKADEV STUDIO
GAME PROGRAMMER
TECHNICAL TEST

I. Written Test

1. OOP atau *Object-Oriented Programming* adalah sebuah paradigma pemrograman yang berorientasi terhadap objek di mana objek merupakan representasi dari sebuah entitas di dunia nyata. OOP berpusat pada objek yang merupakan sebuah instansi dari sebuah kelas yang memiliki atribut dan metodenya masing-masing. OOP dapat membantu program menjadi lebih modular, efisien dan mudah dikelola.
   OOP memiliki 4 prinsip, yaitu *abstraction, encapsulation, inheritance,* dan *polymorphism*. *Abstraction* merupakan prinsip untuk mengabstraksi atau menggeneralisasi sebuah objek sesuai dengan kebutuhannya dengan tujuan memberikan ide atau konsep awal. *Encapsulation* adalah prinsip untuk menyembunyikan implementasi dengan melimitasi aksespada metode dan variabel kelas. *Inheritance* dalah konsep di mana sebuah kelas dapat mewarisi atribut dan metode dari kelas lain sehingga terbentuk hierarki kelas. *Polymorphism* adalah prinsip untuk memungkinkan objek dari kelas yang berbeda untuk merespons dengan cara yang berbeda terhadap panggilan metode yang sama (*overriding* dan *overloading*).

2. MVC atau *Model View Controller* adalah sebuah pola arsitektur yang membagi program mejadi 3 bagian, yaitu *Model* yang bertugas mengelola data, *View* yang bertugas menampilkan data, dan *Controller* yang bertugas menghubungkan *Model* dan *View*. **Contoh**: Sebuah kelas yang bertujuan untuk mengambil dan menyimpan data kumpulan karakter pada sebuah resource file berperan sebagai *Model*. Kemudian kelas UI yang bertujuan menampilkan data-data karakter seperti HP, skills, ATK, dll ke tampilan layar UI berperan sebagai *View*. Dan kelas yang bertujuan untuk menghubungkan data yang diambil dari Model dan menampung logic yang perlu diproses berperan sebagai *Controller.*

3. SOLID Principles:
   a. Single Responsibility Principle: Satu kelas hanya memiliki satu tanggung jawab atau tujuan.
      Contoh: Kelas Kendaraan yang memiki sub-class Mobil, Motor, dll untuk merepresentasikan satu buah kendaraan khusus dan tidak semua sekaligus.
   b. Open-Closed Principle: Sebuah entitas hanya boleh di-*extend* namun tidak dimodifikasi.
      Contoh: Sebuah kelas yang bertugas menyimpan sebuah data ke sebuah txt file kemudian perlu juga menyimpan ke csv file. Maka solusinya adalah mejadikan kelas tersebut sebagai interface yang dapat diimplementasi 2 kelas baru yang dapat mengimplementasi masing-masing tujuan.
   c. Liskov Substitution Principle: Sebuah *instance* dari *sub-class* harus dapat mengsubstitusi untuk *parent class*-nya

Contoh: Kelas Triangle dapat dianggap sebagai kelas Shape karena merupakan turunan dari kelas Shape tersebut.

d. Interface Segregation Principle: Membuat interface yang kecil untuk tujuan khusus dan tidak interface besar yang mengandung banyak tujuan.
Contoh: Membuat interface CanFly, CanRun, CanSwim, Mamalia, Vertebrata, dll untuk dapat diimplementasi pada kelas-kelas hewan dibandingkan membuat satu interface hewan.

e. Dependency Inversion Principle: Modul yang lebih tinggi tidak boleh bergantung kepada modul yang lebih rendah
Contoh: Tidak membuat kelas Profile yang bergantung pada kelas Database karena dapat menimbulkan masalah ketika terjadi perubahan pada Database.

4. *State Machine* adalah model pemrograman untuk memisahkan *state* atau kondisi dari sebuah entitas secara terpisah di mana setiap *state* memiliki prakondisi dan behaviour masing-masing dan transisinya dikendailkan oleh *state machine*.

5. *Object pooling* adalah sebuah metode untuk mengumpulkan sejumlah objek agar tidak perlu membuat objek baru setiap kali dibutuhkan. *Object pooling* baik digunakan saat kondisi dimana perlu dibuat banyak objek berkali-kali.

6. Node, Object, dan RefCounted:
   a. **Node**: Elemen utama yang digunakan untuk membangun scene di dalam Godot. Setiap node memiliki properti masing-masing bergantung dengan jenisnya di mana turunan dari sebuah node pasti memiliki properti node asal turunannya. Sebuah node dapat memiliki *child* dan *parent* sehingga membentuk hierarki.
   b. **Object**: Kelas basis atau paling dasar dari semua kelas di Godot. Objek memiliki properti dan fungsionalitas memory management yang digunakan sebagai basis untuk setiap kelas di Godot.
   c. **RefCounted**: Kelas yang berperan sebagai sistem *memory management* di dalam Godot. RefCounted digunakan untuk menghitung jumlah reference dari sebuah objek yang telah dibuat dan melepasnya ketika tidak lg digunakan.

7. Particle System:
   a. **CPUParticles**: Sistem partikel pada Godot yang diproses menggunakan CPU. CPUParticles biasa digunakan pada kasus dibutuhkan partikel yang relatif sedikit dan fleksibel untuk dimanipulasi.
   b. **GPUParticles**: Sistem partikel pada Godot yang diproses menggunakan GPU. GPUParticles biasa digunakan pada kasus dibutuhkan partikel yang volumenya relatif besar dan performa tinggi

8. Anchor dan Pivot:
   a. **Anchor:** Menentukan titik acuan untuk posisi relatif sebuah elemen terhadap *parent*-nya

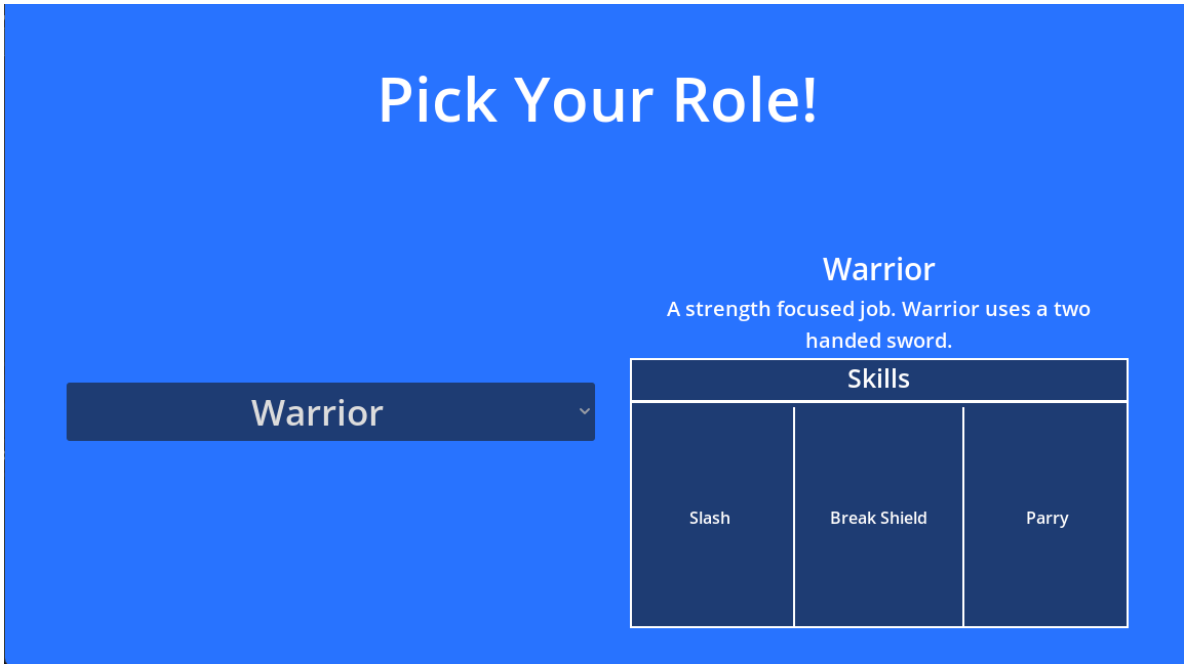
b. **Pivot:** Menentukan titik acuan untuk transformasi seperti *scaling* dan *rotation* sebuah elemen relatif terhadap *parent*-nya

9. Kasus lag atau freeze pada saat transisi ke sebuah scene bisa terjadi karena masalah pada *memory management.* Beberapa solusi adalah dengan melepaskan beberapa resource dari scene-scene sebelumnya yang sudah tidak lagi digunakan untuk meringankan proses. Solusi lain adalah dengan melakukan preloading dan melakukan pooling terhadap resource yang sering digunakan sehingga tidak membutuhkan waktu yang lebih dari menginisiasi banyak resource. Kesimpulannya adalah optimasi kode dan resource agar meringankan dan mengurangi beban proses yang terjadi selama transisi. Dan untuk memberikan UX yang baik, dalam proses transisi ditampilkan *loading screen* sehingga user mendapatkan visual feedback.

10. Pertama membuat sebuah fungsi matematis untuk mementukan exp yang dibutuhkan pada setiap level untuk naik ke level berikutnya kemudian buat atribut dan metode yang relevan pada karakter seperti *gain_exp*, *level_up*, dll. Apabila karakter yang bisa level up lebih dari satu maka dijadikan seperti interface sehingga dapat digunakan pada banyak karakter atau dapat menjadi kelas yang diturunkan. Setiap kali sebuah karakter melakukan aksi yang bisa menambahkan expirience, fungsi *gain_exp* dapat dipanggil baik dari karakter itu sendiri atau memberikan signal kepada karakter untuk menggunakan fungsi tersebut dan diakhir menambah expirience akan selalu dicek apakah sudah cukup untuk level up atau tidak, jika ya maka level karakter tersebut bertambah dan exp yang berlebih diberikan pada leve berikutnya.

II. Coding Test

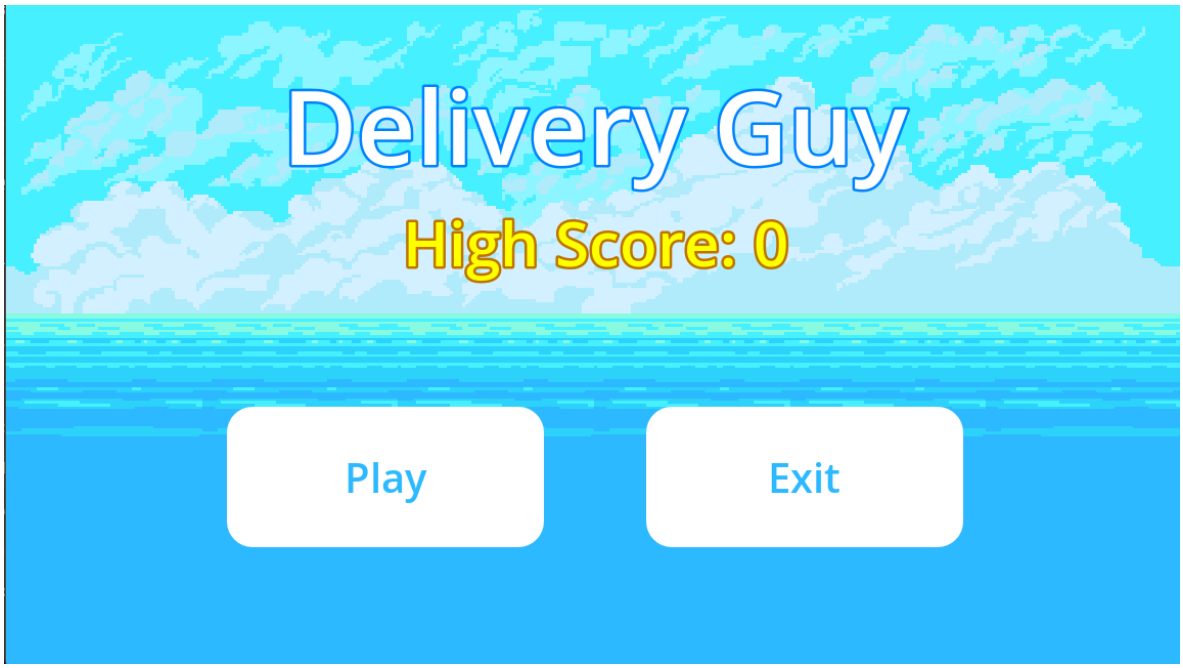
1. **Role Picker with Data From CSV File**

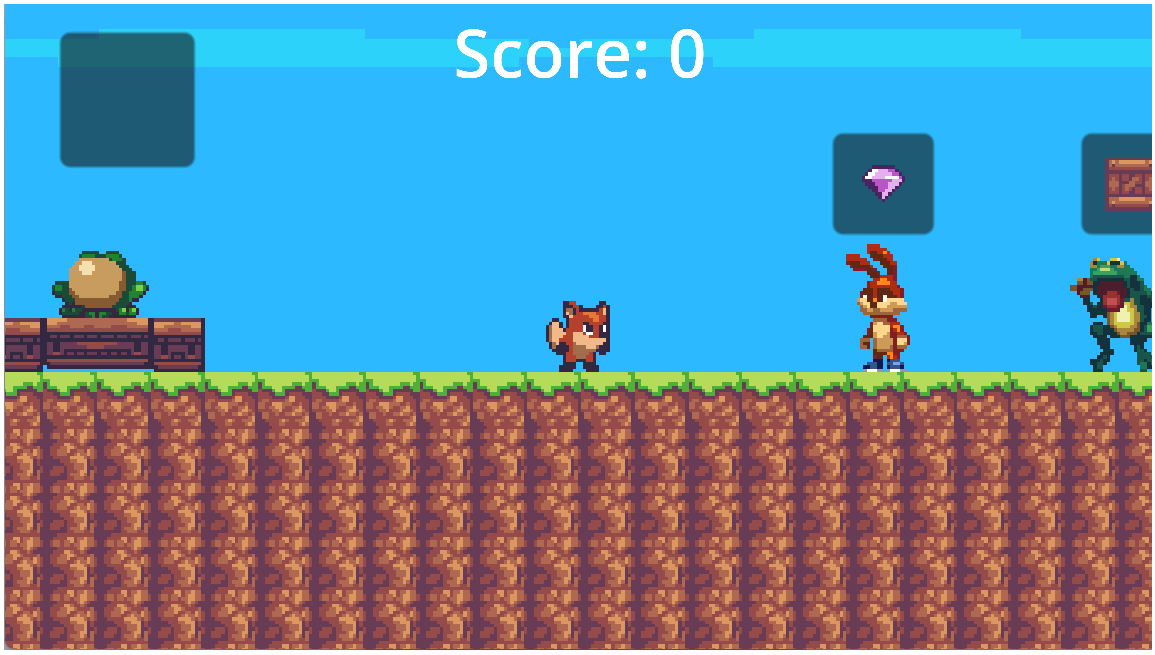


How to run:

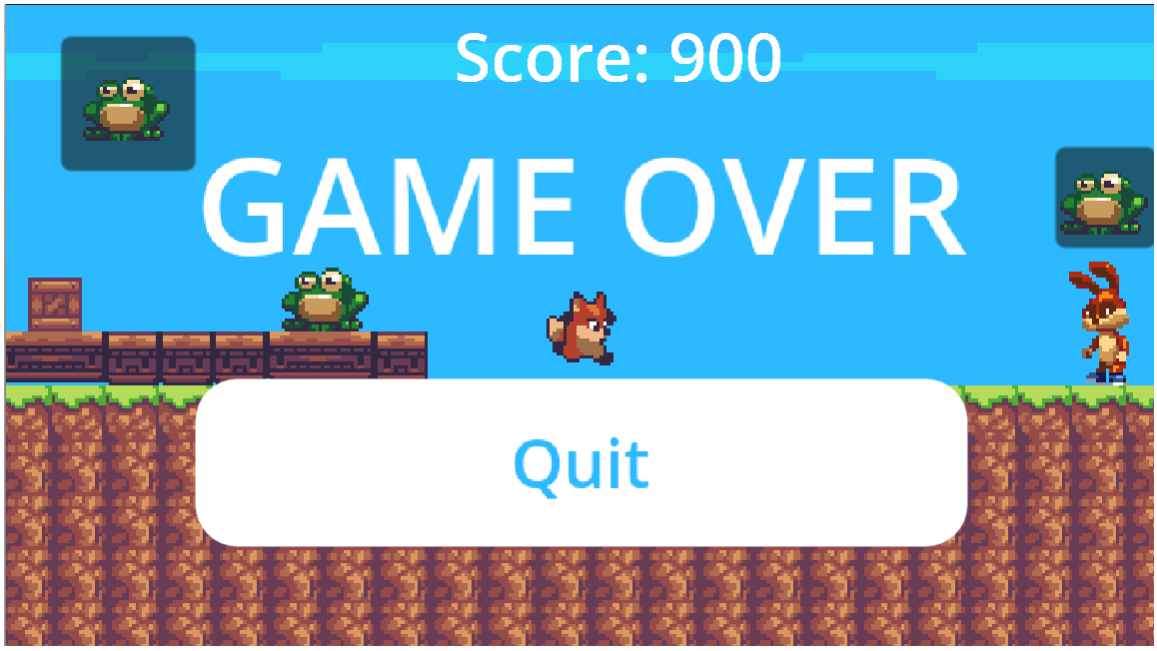
Run `Pick Job.exe` in the `Builds` directory

2. **Delivery Game**




Score: 0


Score: 100
Resume
Quit



How to run:

Run `Delivery Guy.exe` in the `Builds` directory

How to play:

Movement **A** - Move left **D** - Move right **W** - Jump **Action** **K** - Pick Up item / Give Item

Lampiran

Link Github Profile:
https://github.com/Ezaaan

Link Repository Test:
https://github.com/Ezaaan/Ikadev-Studio-Internship-Test

Link Previous Game Projects:

- **Echoes of the Parted**
  https://ezaaan.itch.io/echoes-of-the-parted

- **Komiskom**
  https://arcantica.itch.io/komiskom

- **Sacrificium Optivus**
  https://lompat-pake-w.itch.io/sacrificium-optivus