INSTINCT

GameDocument

Game Overview

Title: Instinct
Genre: 2D survival
Platform: PC Standalone
Target: Casual Gamer

Instinct is a 2D survival game where the player is a robot called Miran who has been trapped in a chamber with blocks spawning randomly. Miran has to dodge blocks that cause him trouble and collide with blocks that help him to survive.

Synopsis

Miran is the young robot in the robot world who is destined to become the ruler. Evil forces try to get rid of him by trapping him in a deadly chamber. Miran has to survive till the end to escape from the chamber and kill the evil forces.

Game Objectives

The objective of the game is to survive till the end and escape from the chamber.

Game Rules

The game is set in a closed chamber so player will not be able to go out of the chamber until game is completed. Player has to move inside the chamber where blocks spawn randomly. There will be three kinds of blocks spawning inside the chamber red, blue and white.

On Collision with red block player dies immediately

On Collision with blue block player size increases

On Collision with white block player size decreases

So, player should dodge blue and red blocks and collide with white blocks.

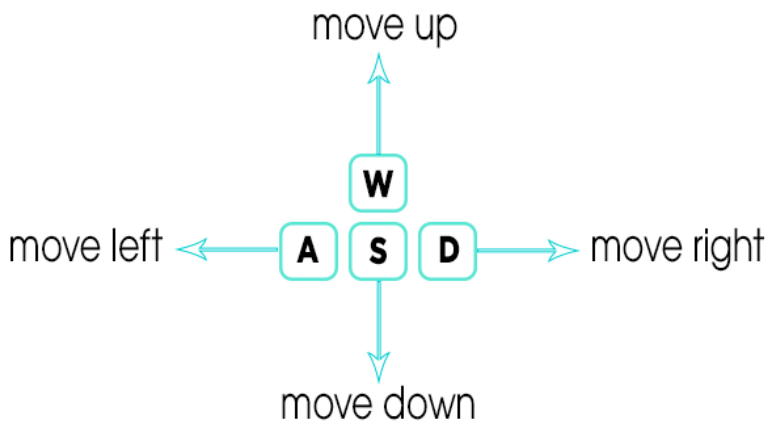
Scoring and Levels

Player score starts at zero when the game starts. To gain score, player has to collide with white block on his lowest size. For each such collision the score increases by one. Whenever the player collides with blue block on his maximum size or collides with boundary walls the score decreases by one.

Levels change based on the score. There are 4 levels in the game. For each level, the following things will change.

- The Map (background of the game will change).
- Number of blocks spawning.
- The speed at which the blocks move.

Game Controls

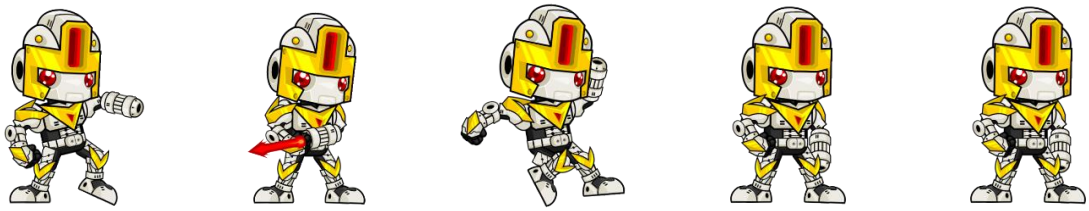


Press W-A-S-D keys to move Miran inside the chamber.
Press Esc key to quit the game.
Press P key to pause the game any time.
Press C key to continue paused game.
Press R key to restart game.

Player

Miran is a young precocious robot in the robot world. He is full of fun and joy.

Player Art:



Player States:

Idle: The idle state is the starting state where an animation of Miran standing still with his body moving is rendered.

Run: Run state is a cycled animation when Miran is moving.

Dead: When Miran is dead an animation of Miran falling forward is played.

Technical Details

This game is built using C++ programming language and SDL.

SDL (Simple DirectMedia Layer):

SDL is a library using which developers can access computer multimedia hardware components. SDL manages video, audio, input devices, CD-ROM, threads, networking and timers.

As part of this project an architectural pattern similar to Entity-Component-System was built to create and manage entities and their components.

Entity-Component-System:

Entity-Component-System is an architectural pattern used in game development.
ECS makes it easy to define and manage entities in a game. Every object in a game's scene is an entity (ex: player, projectiles, vehicles etc). Every entity consists of one or more components which add functionality to entity (ex: A physics component adds physics behaviour like reacting to forces to an entity).
Entity: Entity is any general object in the game.

Component: Component is the raw data for one aspect of the object, and how it interacts with the world.

System: Each System runs continuously and performs global actions on every Entity that possesses a Component of the same aspect as that system.
Ex: If there is a Collision-System then that system performs actions on Entities that have a Collider-Component attached to them.

Systems are not present in the architectural pattern developed as part of this project so it's just an **Entity-Component** pattern.
The pattern is in EntityComponentSystem.h file. It consists of three classes

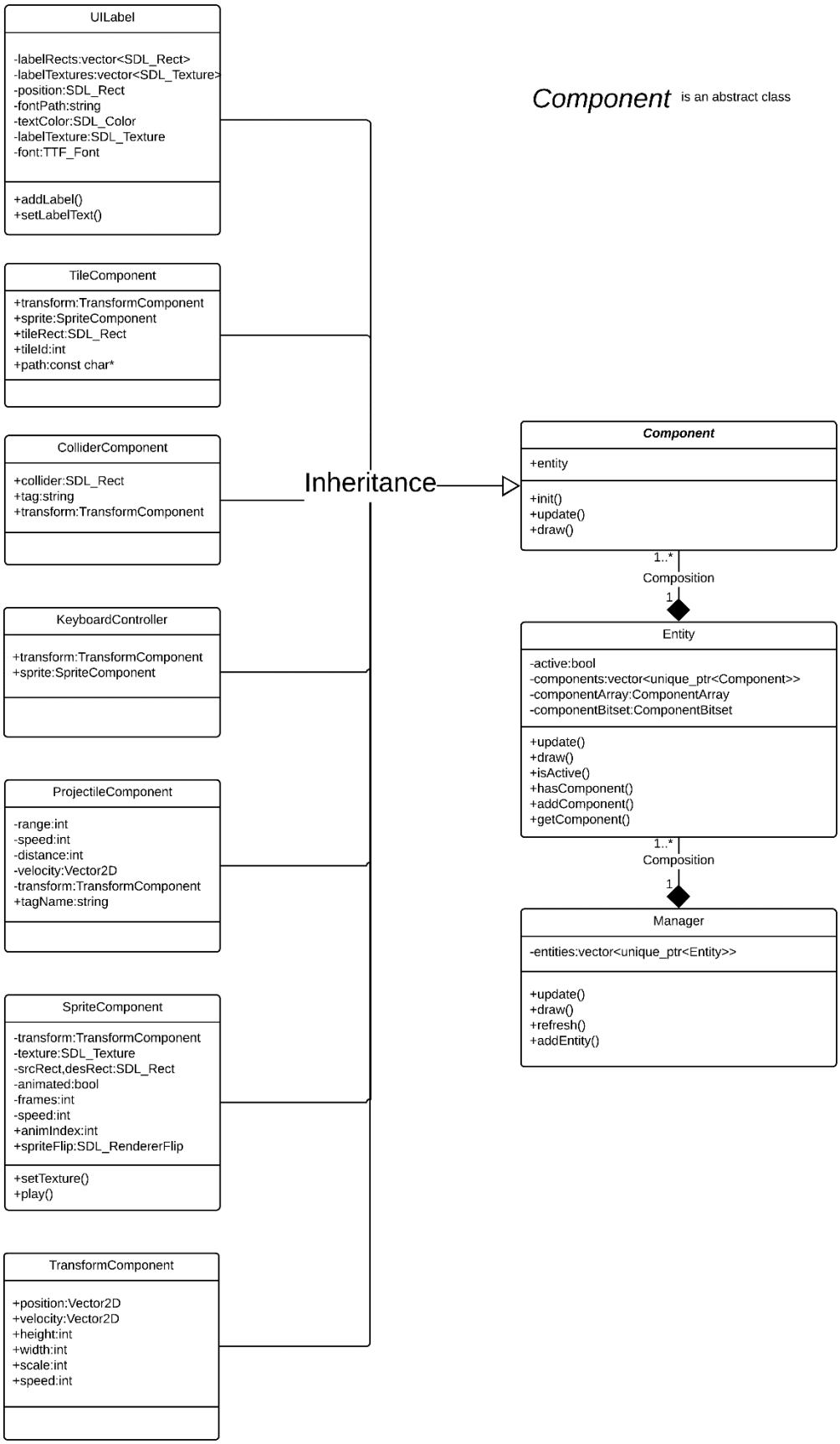
1. Component Class
2. Entity Class
3. Manager Class

Component class is an abstract class and also the base class for specific component classes like TransformComponent class which holds position and scale of an entity and other specific component classes.

Entity class has methods to add components and get them whenever needed. Components add behaviour to entity and they can be updated to change an entity's behaviour. Components of an entity are stored in a data structure.

Manager class adds and manages entities in the game, single manager updates multiple entities. It has methods to add, update, draw and refresh entities. A data structure is used to hold the list of entities of a manager.

UML Class Diagram of Entity-Component Pattern:



Code Flow Diagram (Files):

Code Flow Diagram gives a brief idea about what each file contains and what does it do.