Project Proposal

Sandbox: 2D Game Engine

Daniel Tal (dt2479)	[Manager]
Martin Fagerhus (mf2967)	[Language Guru]
Abhijeet Mehrotra (am4586)	[System Architect]
Roy Prigat (rp2719)	[Tester]

1. Language description

Sandbox is a language that will be geared towards creating a two-dimensional grid-like game world. Properties of the world will be defined by the programmer which include, but are not restricted to, dimensions of the world and dynamic or static objects/obstacles with programmable properties (for ex. hitting an object impacts player life). Conditional event functionality is programmable, this will allow the programmer to define outcomes of the player's interaction with the world. This will create, what can be defined as, a game engine which contains the properties and enforces the rules of the game during play-time.

This can be seen as a foundational layer for easy implementation of a functional two-dimensional grid-like game and simultaneously provides opportunity for more complex games with a multitude of features. Our language will simplify game creation by abstracting away the need for the programmer to account for threading and multiple other lower-level features that are imperative for creating games in other languages. The game developer can therefore spend more time on creative aspects of the game and less time on implementing tedious back-end functionality.

Our language will provide a shell-like log interface which the player will be able to interact with in a command line interface. The log(game status) will be updated based on player interaction and events triggered on a timer basis as the game goes on. This log based game information will be a game engine and will hook into a GUI written in another higher-level language such as Java.

2. Features

- Generate 2D maps, define the walls/boundaries, obstacles and moving elements.
- Define Player entities: location and other starting configurations
- Apply the rules of physics to the world
- Define events:
  - Winning condition
  - Player and obstacle collisions
  - Shooting directions(hitting another player)

3. Data types/structures

3.1 Primitives

Type	Description
int	Integer
float	32-bit floating point number
bool	True/False
char	Character

3.2 Structures

Type	Syntax	Description
Tuple	(int,int)	Primarily used for coordinate representation

4. Conditionals

Keyword	Description	Syntax
if	if statement	if [condition] {...}
else	else statement	else {...}
for	for loop	for[condition] {...}
while	while loop	while[condition] {...}

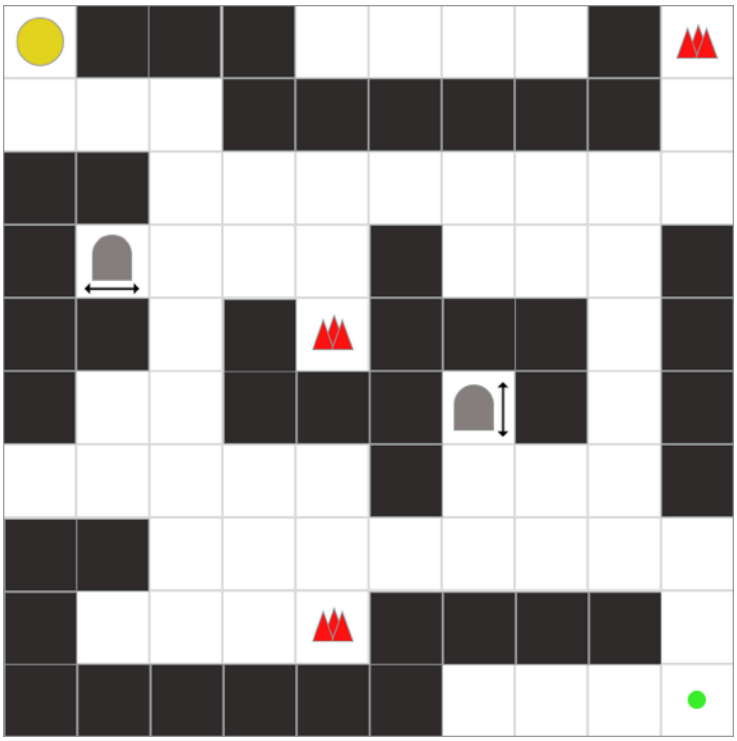
5. Reserved Keywords

Type	Description
World	world is defining the layout of the game. It is the main entity which owns the static and dynamic obstacles and the players that exist in the world. It also owns events like world.begin() and world.end()
Reset	reset the world to its original configurations
Event	signifies event type
Condition	Signifies condition for event action to take place. Defined by event
Action	Signifies event action, defined by event
Player	It is class type of a player object. It has required properties like: initial_position, thickness, bounce, health, lives.
static_obstacle	Is immutable in the world. Is rendered at world_start and doesn't change.
dynamic_obstacle	Is mutable. Can have health associated with destruction, if health=-1, indestructible. It has required properties like: initial_position, thickness, bounce, health. Can also have a speed vector.

6. Operators

Type	Operator	Description
Assignment	=	Right hand value assigned to left hand
Arithmetic	+ - / * %	Addition Subtraction Division Multiplication Modulo
Comments	#	Single line commenting
Boolean	&& ||	AND OR
Relation and Comparison	!= < > <= >=	

7. Sample Program



```
# defining a player(inbuilt type)
player ash {
        position = (0,0);  #initial place on grid when world renders
        thickness = 1;  #width of player
        health = 100;  #health of the player
        lives = 3;  #lives of the player
        speed = 1; #in blocks/second, triggered when the mouse is moved
        event die () {
            condition {
                this.health = 0;
                this.lives = this.lives-1;
            }
        }
            action {
                world.reset();
            }
}
```

```
element static_obstacle wall {
        width = 1;
        direction = 0;
        length = 6;
        damage = 0;
}

element dynamic_obstacle guard {
        width= 1;
        length = 1;
        speed = 1; # in blocks/second
        bounce = true;
        damage = 100;
        health = -1; # infinity
}

event win (player p) {
        condition {
            p.position = (9, 9)
          }
        action {
            world.end();
          }
}

# defining a world with 1 player, a guard that oscillates between the wall and the right boundary.
world w {
        dimensions d = (10,10) #dimensions of the grid
        player p1 = new ash();
        wall w1 = new wall(position = (3,1));
        # guard that bounces between the wall and the right boundary with speed 1.
        guard g1 = new guard(initial_position = (1,3), direction = 0);
        # It had 100 damage and kills the player instantly
        event e = new event(win p1)
}
```