AMRITA SCHOOL OF AI, BENGALURU

B. Tech in AIE/AID

Second Semester, Section G, Academic Year: 2024-25

ELEMENTS OF COMPUTING - 2 (23AID113)

PROJECT

on

Jump Quest

Presented by

Rtamanyu N J (BL.AI.U4AID24060)

Nagam Chaturya Reddy (BL.AI.U4AID24047)

Chandana Srinivasan (BL.AI.U4AID24016)

Gokul Krishna (BL.AI.U4AID24024)

Overview

- This game falls under Arcade Genre
- The game consist of moving platforms, which change direction at random (the gravity also changes).
- The bottom of the screen is the void, where if the player falls it is game over.
- There is a day and night system which is activated periodically.
- JumpShift is a 16-bit endless runner where the world defies your existence, standing against you pushing you to give up on chasing the stars. Dodge obstacles, jump with precision and reshape the very core of the game's world to your advantage. Master the art of gravity manipulation to defy all odds and survive as long as you can.

Overview

- The player has 5 abilities: Jump, Dash, One time Save and Time Delay.
    - Jump – Lets the player jump to next platform
    - Dash – Lets the player move a bit forward fast.
    - Stabilize – the player can stop falling to the void one time in his playthrough.
    - Time Dilation – the player can slow down time for better grasping of the scene
    - Gravity swap – Allows the player to swap gravity.
- All the platforms are procedurally generated using persistent rendering method for faster rendering.

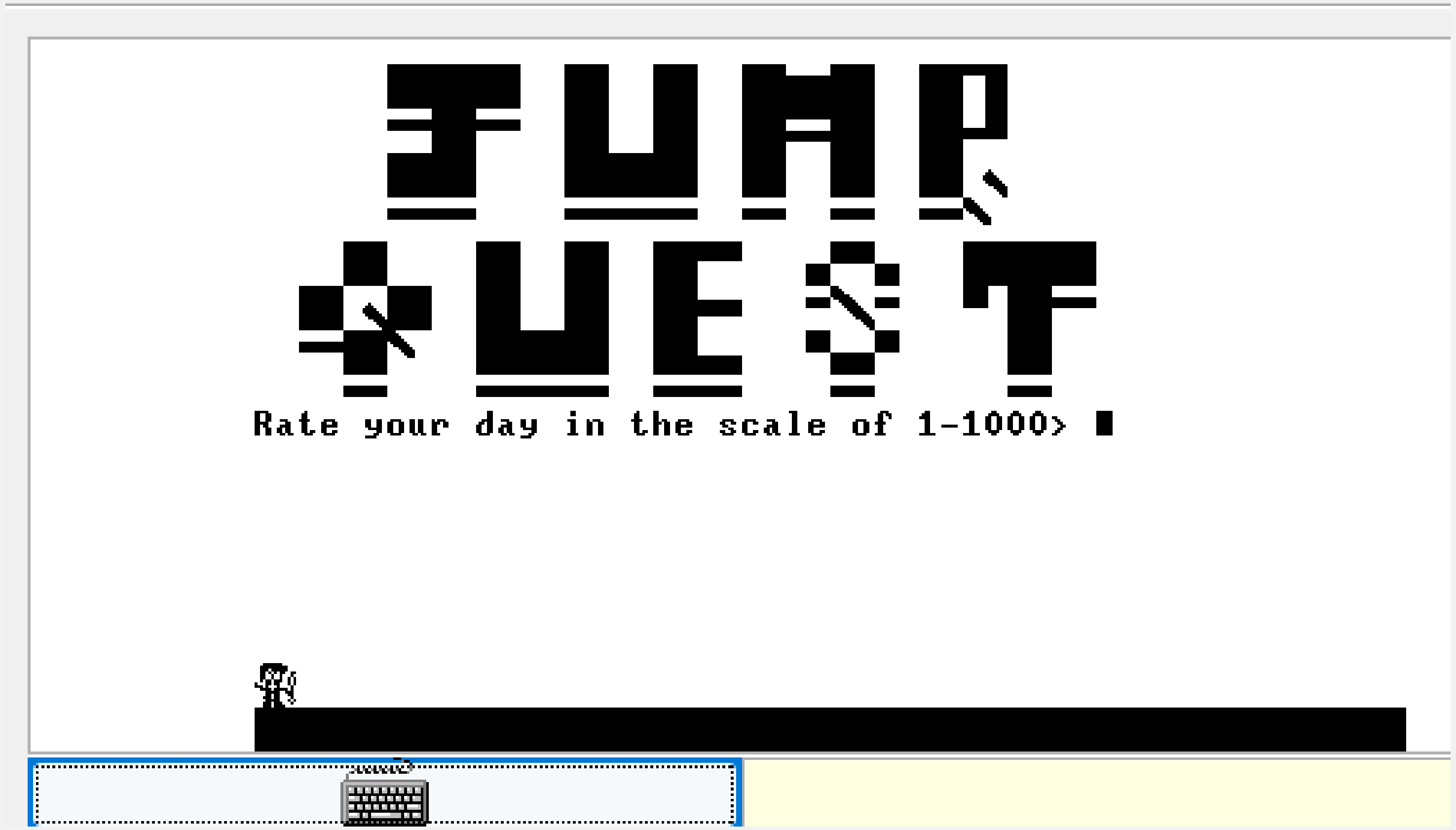
Overview → Backstory

- The game starts with a man running away holding a sword in this right hand. He jumps from objects to objects escaping an unknown shadowy entity. This entity however is not visible to player himself. The game progresses as you keep running away from an entity that you do not see yourself.

- The man is named Claid and he is running relentlessly showcasing his proficiency in running and parkour. He falls many times but he feels no pain doesn't die but the shadowy entity eliminates him. Does he die? No. He simply begins again. Initially he asked himself, "Why am I not dying? Was that a dream?" before he starts running away from the shadowy figure again but over the span of time he has given up. Given up looking for answers about this mysterious world where death and pain doesn't exist.

- "I am running… running away." that was the last thing in his memory. "But from what?", His throat was dry from the same question, yet no answer ever reached it.

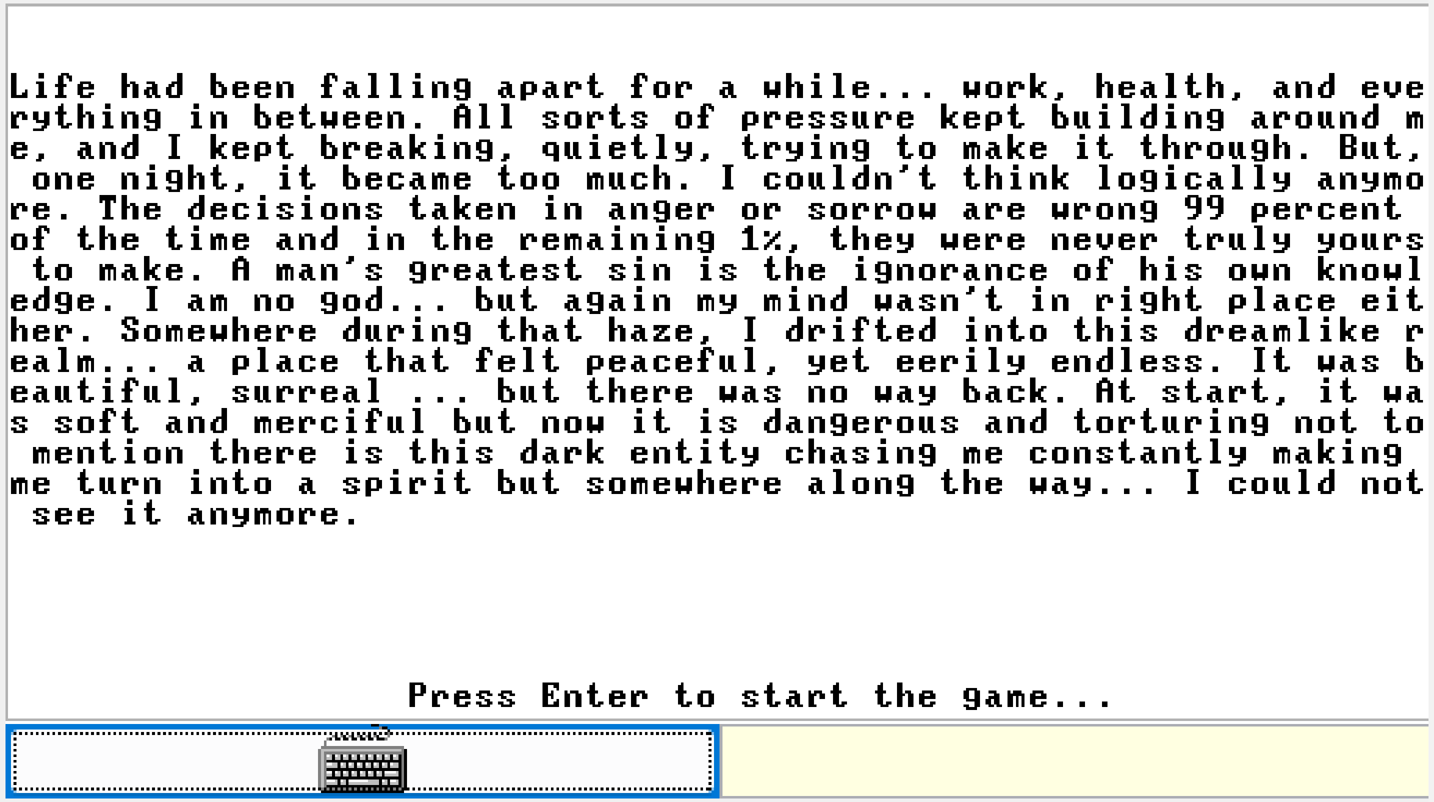
Game Layout



```
Rate your day in the scale of 1-1000>
```

Fig. 1: Game Play (Starting scene)

Game Layout



```
Life had been falling apart for a while... work, health, and eve
rything in between. All sorts of pressure kept building around m
e, and I kept breaking, quietly, trying to make it through. But,
 one night, it became too much. I couldn't think logically anymo
re. The decisions taken in anger or sorrow are wrong 99 percent
of the time and in the remaining 1%, they were never truly yours
 to make. A man's greatest sin is the ignorance of his own knowl
edge. I am no god... but again my mind wasn't in right place eit
her. Somewhere during that haze, I drifted into this dreamlike r
ealm... a place that felt peaceful, yet eerily endless. It was b
eautiful, surreal ... but there was no way back. At start, it wa
s soft and merciful but now it is dangerous and torturing not to
 mention there is this dark entity chasing me constantly making
me turn into a spirit but somewhere along the way... I could not
 see it anymore.

                  Press Enter to start the game...
```

Fig. 2: Game Play (Back Story scene)

Game Layout

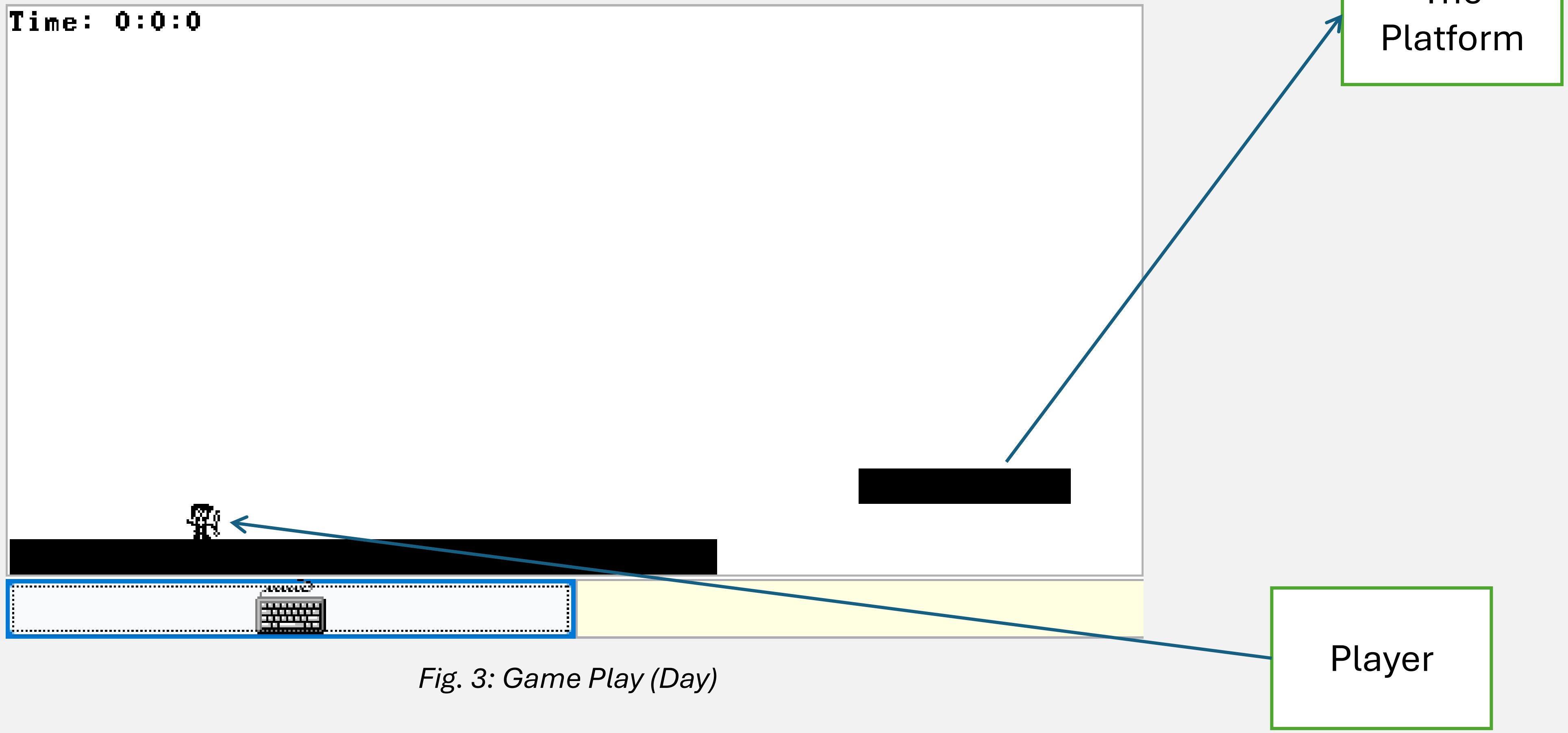


Fig. 3: Game Play (Day)

Game Layout

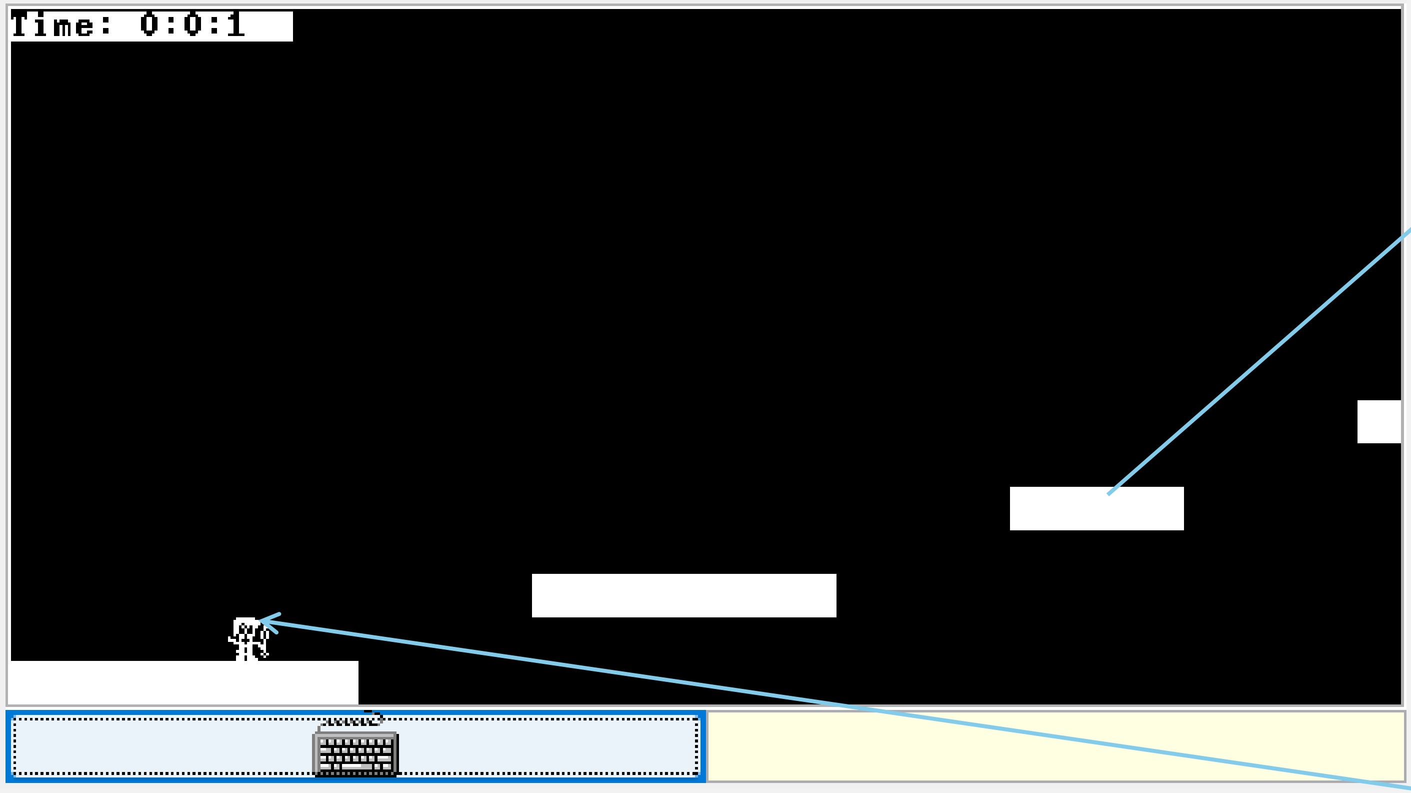
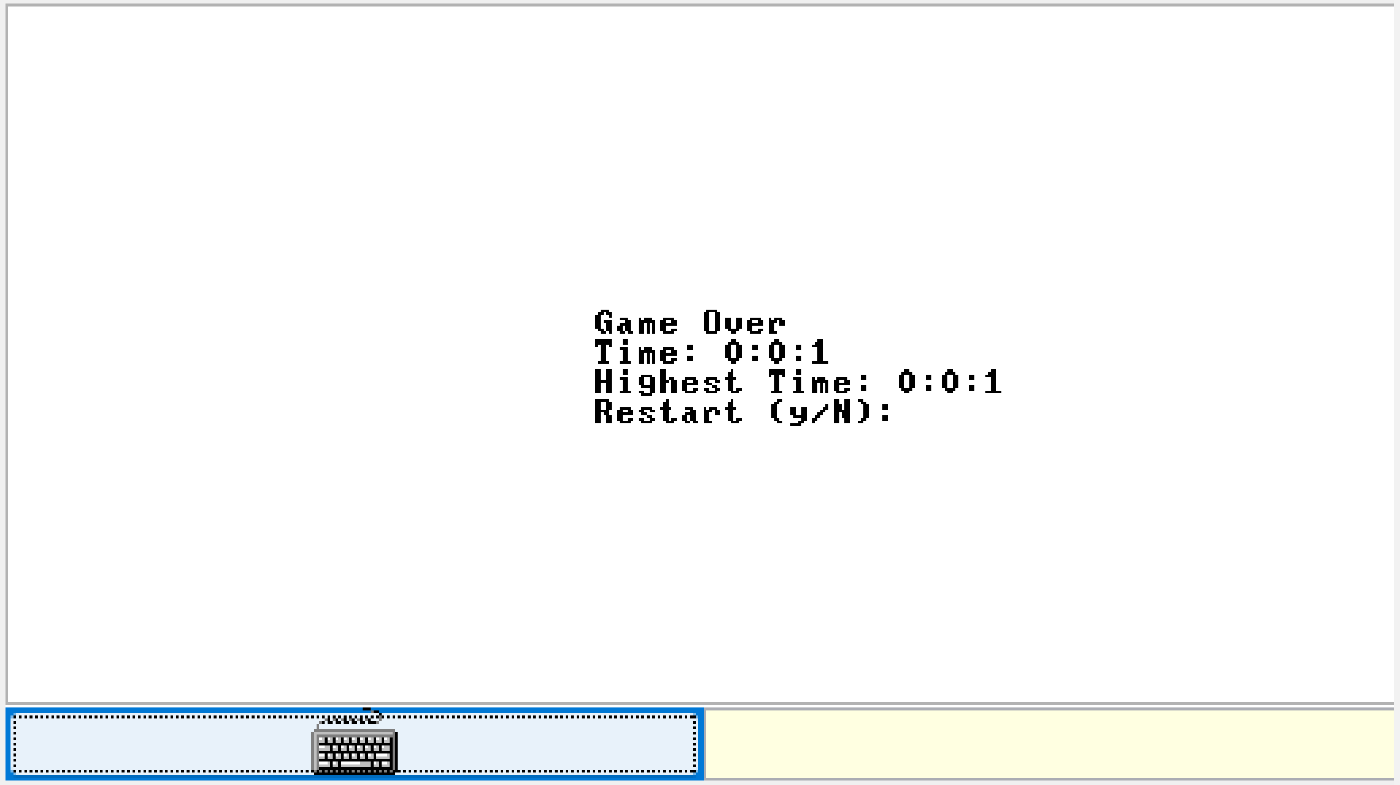


Fig. 4: Game Play (night)

Game Layout



```
Game Over
Time: 0:0:1
Highest Time: 0:0:1
Restart (y/N):
```

Fig. 4: Game Play (Game Over Scene)

How to play

- The start of the game player must input a random number and hour and minute in 24-hour format.
- The player can press spacebar key for jumping
- The player can press ‘F’ key for dashing
- The player can press ‘R’ key for one time save (it will only work one time).
- The Player can press ‘G’ key to delay time.
- The player must take care of not going out of screen bounds as it will lead to the end of the game.

File details

Main.jack:

- Initializes game controller
- Starts the game.

GameController.jack:

- The main brain of the game.
- Delegates task of platform, background, player and ground.
- Responsible for change in gravity, day/night, platform direction, Score maintainance, procedural platform creation and enabling player abilities.

Rd.jack:

- Random function made by us
- Creates random no. between two values

Background.jack:

- Responsible for rendering background sprite.
- Responsible for an easter egg (the true ending)

File details

Ground.jack:

- Responsible for triggering void and end game scene

Platform.jack:

- Stores all the platform created
- Renders the platform using persistent rendering method.
- Handles the display of platform according to the direction.

Player.jack:

- Responsible for stimulating time.
- Responsible for drawing and interacting with player.
- Responsible to enforce gravity onto player.
- Responsible for collision systems.
- Responsible for random gravity and player’s abilities.

Flowchart

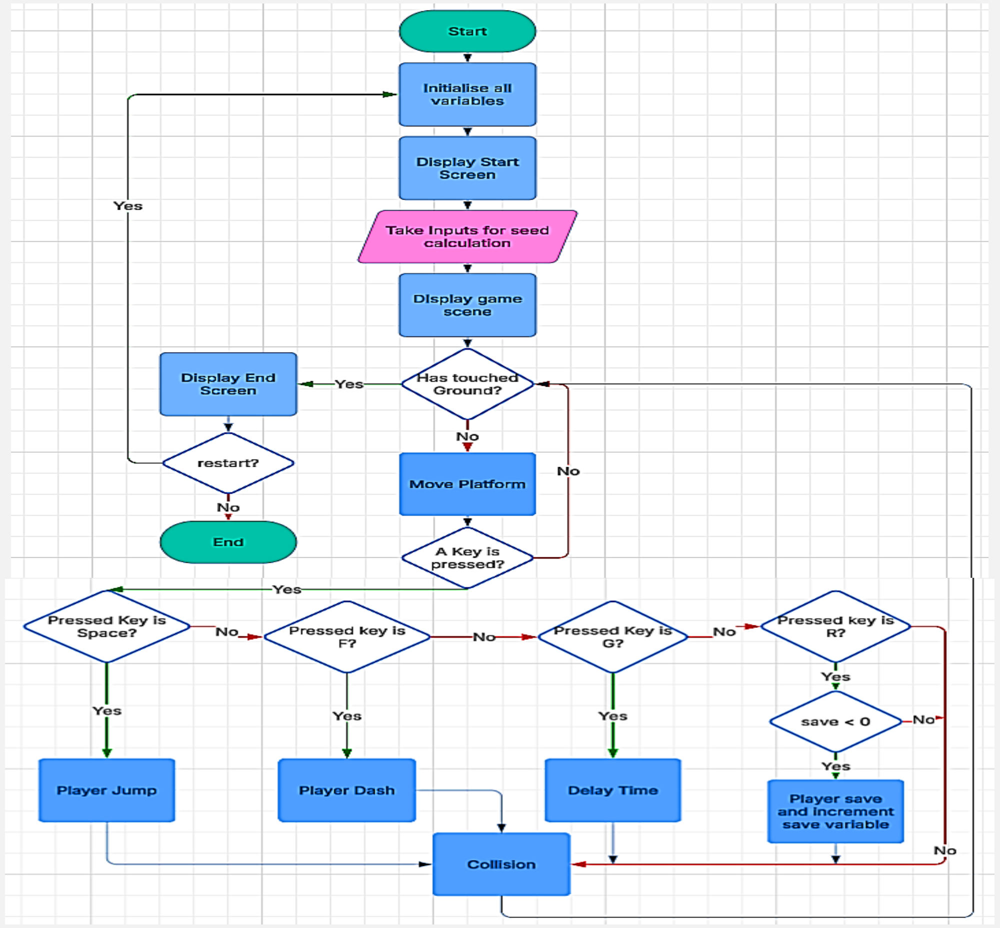


Fig. 5: Flow Chart

Class Diagram

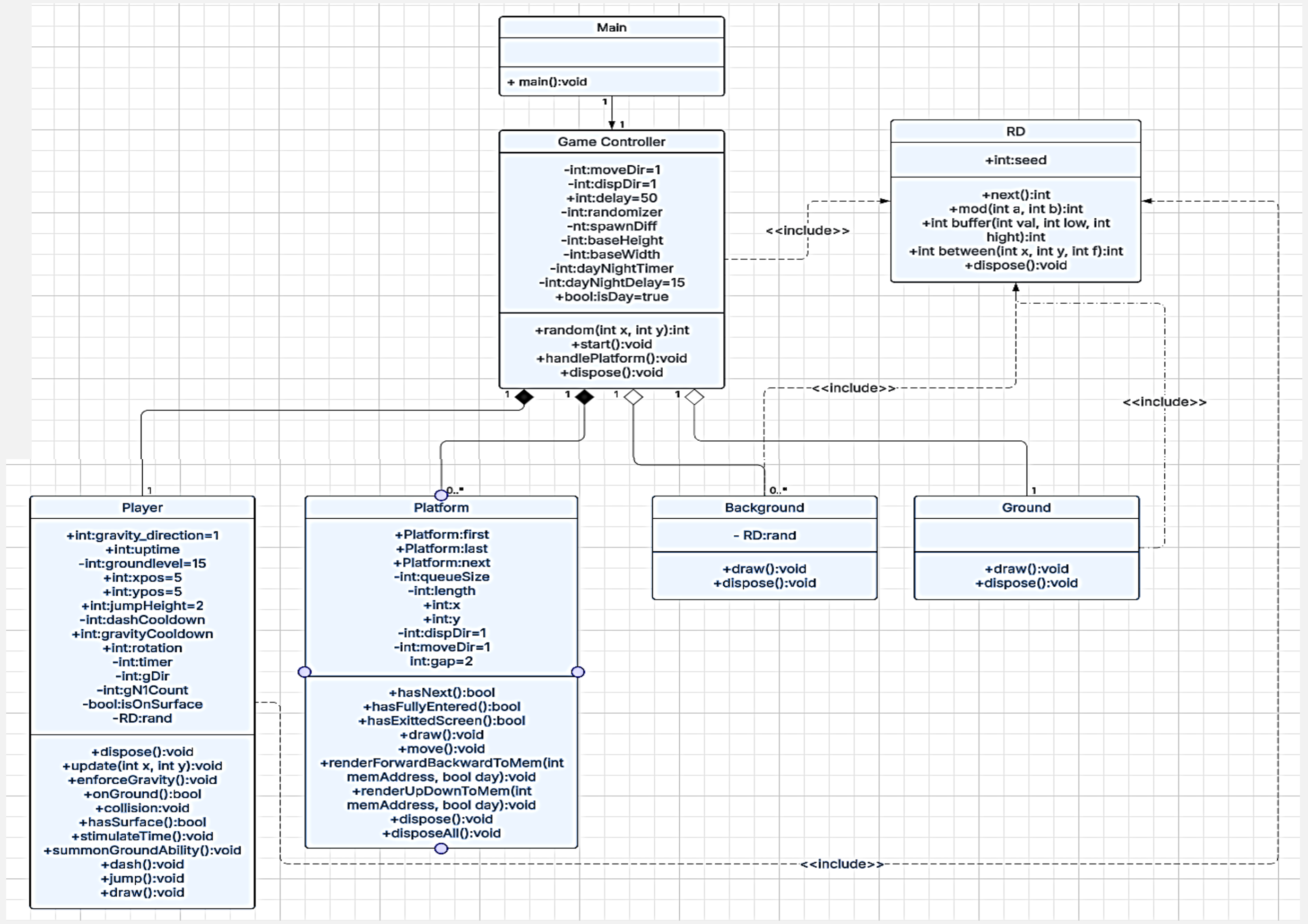


Fig. 4: Class Diagram

Use Case DIagram

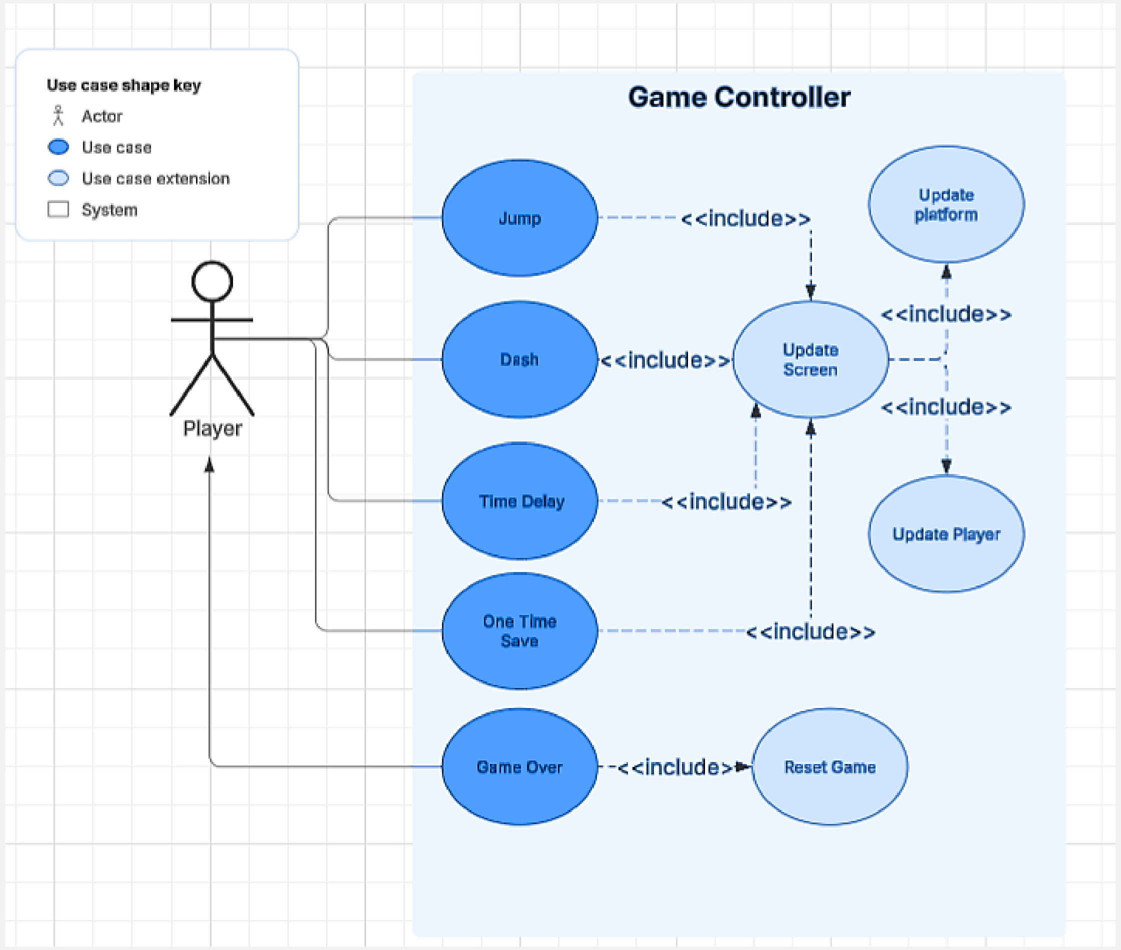


Fig. 5: Use Case Diagram

Objectives

- Random Function
- Procedurally Generated Platforms
- Player
- Score
- Day and night System
- Direction System
- Gravity system
- Collision System
- Rendering System

Implementation details

Implement deque using linked list

- We integrated the features of linked list and deque in the same class Platform.
- 2 static variables (first and last) and 1 local variable next was used to implement the whole idea.
- The Game controller had access to first and last which was used to traverse through the linked list.

Implement tile drawing

- We referred to nand2tetris tutorials[1] to do the basics of tile rendering.
- We used the bitmap editor[2] to draw tile and get corresponding values to put in memory
- We created a x and y coordinate system that is used to locate the position of everything that is rendered on scene.
- We also created code to convert the x and y coordinates to corresponding memory location which can be used by the renderer to put the values of the tiles into memory

Implementation details

Implement persistent rendering

- Just tiling alone increased the no .of draw calls tremendously when rendering the next frame, so a new technique of persistent rendering was introduced.
- In original rendering, the previous tiles were erased and then the next frame tiles were rendered.
- In persistent rendering, only those tiles which had to be changed in the original frame were changed in the next frame leaving the others untouched, this significantly reduced the draw calls hence decreased the lag due to rendering.

Implement day and night

- We created two sets of rendering one for day and one for night.
- We created a timer controlled by the game controller, which changes the mode (day/night) when the timer hits 0.

Implementation details

- **Implement direction change**
  - We checked if the direction of gravity was changed.
  - When it was changed, we called the specific direction change method depending on the gravity direction
  - The direction change function, clears the screen, adds the mode and background, then spawns in the platform as per direction, and finally spawns the player.

Implementation details

- **Implement Start Screen**
  - We created a Starting Screen. The big text spelling the name of the game was rendered directly by tile mapping.
  - We let enter key to mark the start of the game.
  - We ask user to enter a random scale value and time in order for the randomizer to work as well as for deciding the time for the easter to be displayed (This is taken once per gameplay).

Implementation details

- **Implement Score System**
  - In this game the scoring done based on the time the player can jump and run through the game without falling to the void.
  - The high score is updated whenever the player beats his previous record.
- **Implement Game over Screen**
  - The high score and current score are displayed in the game over screen.
  - It is invoked whenever the player falls in the void.

Implementation details

- **Implement Random**
  - Time plays a major role in generating true randomness primarily because time is strictly increasing linearly. Jack doesn't have a method to access system's time. To overcome this a local timer was used and stimulate_time kept track of it.
  - Random class has a seed which will be update by bitwise operators during runtime.
  - The number returned is the buffer of the expression below:
    - mod(mod(seed,range)+(GameController.get_randomizer()+x)+(GameController.get_randomizer() * Player.get_time()) + f, y);
  - The get_randomizer returns a special unique number collected at the start of game from user. Entering the same input may help to get the same playthrough. This is explained soon.
  - Buffer is another method which ensures certain number given lies within specified range.

Implementation details

- **Implement Random → Randomizer (Seed from user input) vs Seed (Constant manipulators)**
    - Randomizer is a number generated from user's input. The input collected is seed and time in real life. However, the Random class is invoked immediately to randomize the collected input. This doesn't ensure fully randomized input but it helps to keep it random. Similar thing is done with the input "time" and their sum is stored in a variable randomizer.
    - Randomizer is used in Random class to manipulate the numbers generated.
    - Seed on the other hand is the constant values stored in each instance of randomizer class to further assist in generating the random numbers.

Implementation details

- **Implement Player**
  - The biggest class in this project is Player. It handles drawing of player, collision system of player and stimulates time.
  - stimulate_time being the most important feature of player class; It is responsible for the following:
    - Updating time in game
    - Swapping gravity on random instance of time generated by Random class. When the variable timer hits 0 then gravity is swapped randomly.
    - It performs collision checks regularly.
    - It reduces abilities cooldowns.
    - It enforces gravity or jumping of player

Implementation details

- **Implement Player → Collision System**
  - The collision system is responsible to decide if player is on a surface or not. This decision has various effects in game:
    - If player is not on any solid surface, then they cannot jump and gravity will drag them down
    - If player strikes (collides) any surface before ground them their current position is made as ground
    - When players walks beyond the solid surface under them, the collision system toggles on ground status of players blocking their jump and enforcing gravity on them

Implementation details

- **Implement Player → Collision System → Logic**
  - Check the pixel right under, above, after and before player depending on the gravity status. If the pixel is having non zero value during day or zero value during night then the player is considered to be on a surface.
  - Constraints:
    - Only apply the checks if player is beyond the default ground level set by gravity directions.

Implementation details

- **Implement Player → Drawing**
  - Bitmap editor helped to design the player model. There are various models of players stored for different use cases like the negation model during night.
  - The direction of player is influenced by rotation variable which is a 4-digit number as: where all 3 r's determine rotation angle and m determines the mode of operation. If m = 0 then normal model is loaded and if m =1 then model is mirrored vertically.
  - The draw also accepts a Boolean argument which helps to load/clear the model.
  - The wiping method and drawing method varies based on day/night.
  - If the time is night, then the negation model is loaded (White and Black pixels are swapped).

Implementation details

- **Implement Player → Drawing → Rotation**
  - All the rotated models are obtained from bitmap editor.
  - As mentioned, the variable rotation is a 4-digit number where first 3 digits (from left) gives the angle and the unit digit provides mode of drawing. If mode = 1 then the image is mirrored vertically.
  - The mirrored vertically is better explained as: “Image of player on the axis perpendicular to the foot of player based on his rotation”
  - The set_rotation(18, mode) allows to set mode and set_rotation(angle, mode) allows to set angle and mode.

Implementation details

- **Implement Player → on_ground, has_surface, jump and enforce_gravity**
  - on_ground: If the player's position is on the pre-determined screen limits based on gravity direction it returns true else false.
  - has_surface: Checks if the player has any means of contact with solid surface be it collision or on ground.
  - enforce_gravity: Clear the player model, update the coordinates based on gravity direction to move the player one step towards ground level and re-draw player. It also effects the on surface status of players.
  - jump: Clear the player model and update the coordinates based on gravity direction to move the player one step away from ground level and re-draw the player. It doesn't effect the on surface status of players. However this makes has_surface return false.

Implementation details

- **Implement Player → Abilities**
  - Dash: The forward movement coordinates of player is increased by 5 units making the player rapidly teleport forward. Has cooldown of 10 seconds IRL.
  - Gravity swap: Allows players to change gravity. This force updates the gravity direction as specified by player. Has cooldown of 30 seconds IRL.
  - Stabilize (To be implemented): Draws a solid platform under player to stop their fall. Can be used once in a playthrough.
  - Time dilation (To be implemented): Increases delay in the game controller per frame. Has a cooldown of 60 seconds IRL.

Implementation details

- **Implement Easter Egg**
  - We have also incorporated an easter egg that marks the end of the game.
  - Using the data collected during the first playthrough of the game a random time is chosen for the easter egg to be accessible to the player.
  - A random key is also decided, which when pressed after the time of activation of the easter egg the player will be able to achieve the true ending of the game.

Challenges Faced

We needed way to procedurally generate platform and efficiently store them dynamically as to not consume lot of memory and not have predefined no. of platforms:

we used random function for generating platform

We used queue implemented with linked list as our datatype for efficient and dynamic storage of platform.

We dispose unused platform when it exits screen, so it saves lot of memory

We faced issue in random function, most of the random functions online did not have the random element after 3 or 4 calls.

We created our custom random function, which randomizes based on the answer of the user for the first 2 questions.

Challenges Faced

We faced issue while rendering, the first iteration of it was very laggy and not responsive enough	• We then used tile-based rendering to decrease lag. • Furthermore, we used persistent rendering for platform to reduce the no. of draw calls required to render the shape.
Issues faced with collision system	• The day/night cycle made the collision system render inefficient during night. The code was updated to reflect this. • The collision system cannot be optimized as one single function due to gravity direction and thus multiple if-else statements were used to take gravity directions into considerations.
Issues with player	• The draw function of player constantly produced player looking in one direction which made a visual inconsistency. Multiple models and conditional draw was used.

Challenges faced

- **Issues with player**
  - The day/night cycle broke the jumping animation of player. Including a negation model fixed this.
  - The player was capable of spamming jump to infinitely jump effectively making player fly. This was fixed with the on surface status
  - The player passed through collision objects before collision registered. This was adjusted by calling the collision system after all computations on gravity were done.

Limitations/Bugs

Player Input is delayed sometimes

Rendering Lags sometimes

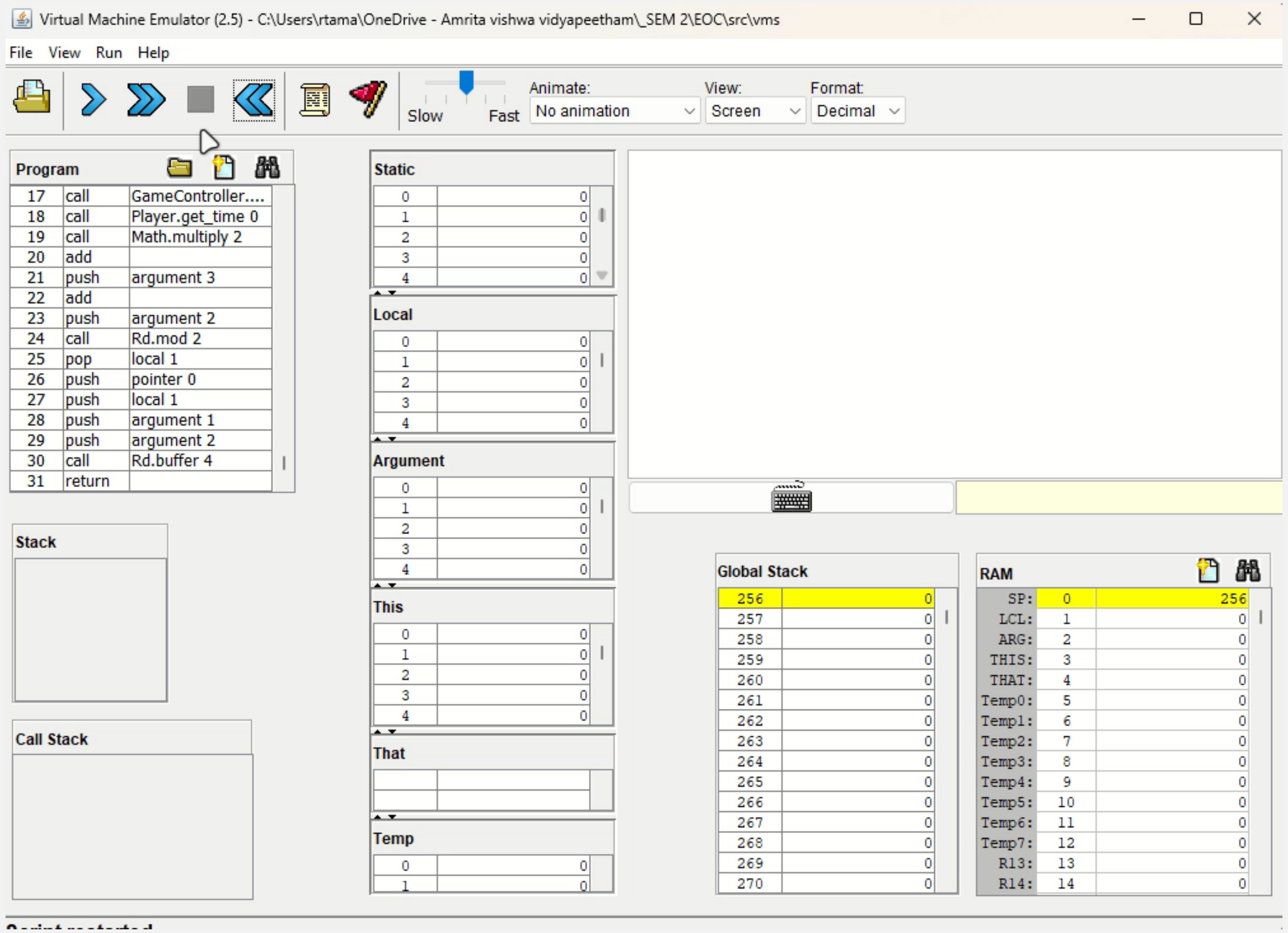
Future Works

ADD CLOUDS AND STARS

ADD BACKGROUND BUILDINGS

IMPLEMENT DIFFICULTY LEVELS

Game Play



Virtual Machine Emulator (2.5) - C:\Users\rtama\OneDrive - Amrita vishwa vidyapeetham_SEM 2\EOC\src\vms

Animate: No animation View: Screen Format: Decimal

Program

```
17  call    GameController....
18  call    Player.get_time 0
19  call    Math.multiply 2
20  add
21  push    argument 3
22  add
23  push    argument 2
24  call    Rd.mod 2
25  pop     local 1
26  push    pointer 0
27  push    local 1
28  push    argument 1
29  push    argument 2
30  call    Rd.buffer 4
31  return
```

Stack

Call Stack

Static

0	0
1	0
2	0
3	0
4	0

Local

0	0
1	0
2	0
3	0
4	0

Argument

0	0
1	0
2	0
3	0
4	0

This

0	0
1	0
2	0
3	0
4	0

That

Temp

0	0
1	0

Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

Video 1: Demo video

References

[1] https://youtube.com/playlist?list=PLNMIACtpT9BfztU0P92qlw8Gd4vxvvfT1&si=KtaeoIcLFV_MlJ3E

[2] Nand2Tetris Bitmap Editor