This is the report for the project “Genetic Algorithm: Doodle Jump” from AMEUR/MENGOZZI

I – Summary

The project consists of 2 main composants: A minimalist Doodle Game type of game, playable by the user as well as by a genetic algorithm. The latter has to improve the performance of a bot each new game.

(VIdeo which inspired our project) : <https://www.youtube.com/watch?v=sB_IGstiWlc>)

II – Description

Doodle jump is a mobile game. The player controls an alien that needs to rise. Thus he needs to bounce from platform to platform. The character bounces automatically and the player only needs to focus on the direction (left, right, stationnary). There’s also ennemies and special platforms, but because of a tight schedule, these features will only be added at the end if we have time. (More details here : <https://fr.wikipedia.org/wiki/Doodle_Jump>).

III-/Game Functions

**Gamemechanics.c**

float random\_float\_generator(float value);

Give a random float between 0 and value

int random\_int\_generator(int min, int max);

Give a random integer in the [min,max] range

PLAY spawn\_player(void)

Initializes player characteristics (starting position, score, arbitrary n° of platforms, time spent mid-air and state (alive or dead))

PLAY move\_player(PLAY player, int keyboard)

Moves the player according to the key pressed and whether or not the player hits the border

void platform\_bounce(PLAY\* player, PLA\*\* platforms\_list)

Manages the jump according to the time spent mid-air and whether the player reached the platform or not

PLAY score\_up(PLAY player, PLA\*\* platforms\_list)

Manages the score of the player and the score given by the platform, if the player jumps on a platform the score is given +1 and the platform loses its ability to give anymore points (scorePlat)

PLAY death\_player(PLAY player)

If the player Y position reaches 0, the player dies

PLA\*\* malloc\_platforms\_list(void)

Allocates memory for platforms, place the first platform and calls initial\_spawn\_platform

PLA\*\* initial\_spawn\_platform(PLA\*\* platforms\_list, int index)

Creates an array of 7 platform structures and gives them a random (x,y) position but taking into account that each platform can be reached from the previous one

void scrolling(PLA\*\* platforms\_list, PLAY\* player)

When the player reaches a certain height, everything scrolls down to allow the other platforms to take place

void check\_platforms(PLA\*\* platforms\_list)

Checks if one platforms is below a certain level to allow it to be replaced

void replace\_platform(PLA\*\* platforms\_list, int previousIndex)

Changes the disappearing platform’s coordinates and scorePlat to optimize the number of platforms

void desalloc\_platforms\_list(PLA\*\* platforms\_list)

Frees the platform when the player quits the game

void draw\_background(void)

Draws background

void draw\_platforms(PLA\*\* platforms\_list)

Draws platform

void draw\_player(PLAY player)

Draws player

void draw\_generation\_score(int indexGenerations, int scoreMax);

Writes score and generation n°

**Gamemechanics.h**

typedef struct PLATFORM //Definition of a platform structure

{

int Xpos; X position

int Ypos; Y position

int scorePlat; Score giveable by the platform

} PLA;

typedef struct PLAYER //Definition of a player structure

{

int Xpos; X position

int Ypos; Y position

int score; Player score

int jump; Platform on which the player can jump on

int jumpTime; time spent mid-air

BOOL alive; whether the player is alive or not

int color[3]; players color (random so we can see all the bots trying at the same time)

} PLAY;

**Genetics.h**

void move\_bot(PLAY\* player, PLA\*\* platforms\_list);

Used to choose randomly left, right or middle

int bot\_eyes(PLAY\* player, PLA\*\* platforms\_list);

Give to the "bot" the user-accessible information of its relative-position from the next platform

PLAY\*\* malloc\_players\_list(void);

used to allocate a list of players

void desalloc\_players\_list(PLAY\*\* players\_list);

used to free the list of players

void regen\_platforms\_list(PLA\*\* platforms\_list);

used to generate a new list of platforms for each iteration

int best\_score(PLAY\*\* players\_list);

used to get the score of the highest player of each iteration

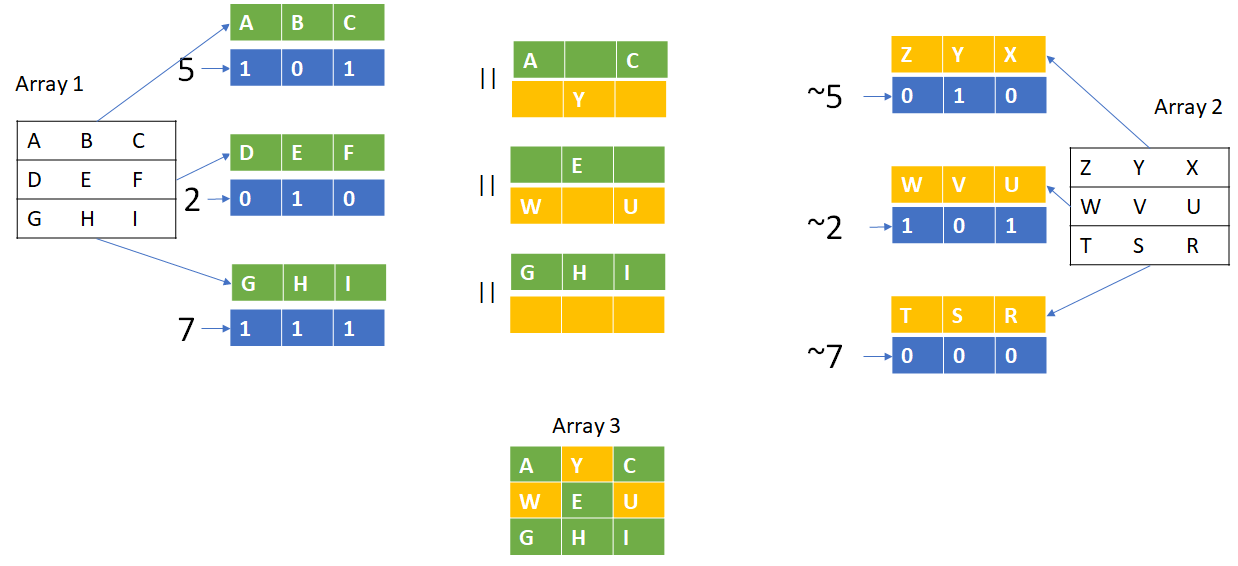
void select\_keyboard(PLAY\* player, int Xrelat);

used to choose the next action of a player (emulates keyboard input)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Left | Don’t move | Right |
| Xrelat<0 | x | y | z |
| Xrelat=0 | a | b | c |
| Xrelat>0 | d | e | f |

PLAY crossover(PLAY\*\* players\_list);

We generate 3 random numbers (here 5,2 and 7 for the sake of comprehension), store their binary equivalent in arrays and make a “if the mask[i][j]==1, then tableau3[i][j]=tableau1[i][j] else tableau3[i][j]=tableau2[i][j]”



void mutation(PLAY\*\* players\_list);

Select a random gene in the genome of a random player in order to change it either radically, slightly or not at all

void natural\_selection(PLAY\*\* players\_list);  
Change the genome of the weakest players with the crossover genome