Jalil : Physics Log

The hardest difficulty was to get the formulas right

By choice, I decided not to include acceleration variables, but only keep speed and positions

after finding our classic equations :

-x = v0 * cos(angle)*t + x0

-y = -0.5*g*t**2 + v0*sin(angle)*t + y0

The biggest problem by far was that the origin (0,0) is situated at the left top corner of the screen, thus

throwing out the window our formula for y

luckily, after some research, this problem can be fixed by multiplying by -1 our formula

(in our movement function, we use the formula of vx and vy as we also alter those with gravity and function)

As for gravity and frictions, we set basic conditions that influence vx and vy:

-Is the ball in air

-Is the ball on the ground and does it move?

17/02 SAT algorithm to detect collisions finished

This version of SAT is working with a circle and a whatever polygon you choose, as long as you specify its vertices

-22/02 frictions on the ground are good and we can "choose v0" by pressing the space bar for a certain amount of time

Now the biggest difficulty by far, is to make all of this work with tiles, including slopes

We want to be able to distinguish between rectangular tiles, triangular tiles, or other shapes, moreover there will be differentiation

with the angle with the floor, the material and things like that

Solution : Give a unique ID for each tile, with condition at the creation

if tile_index in [0,1,2,3,4] angle = 0 and vertices = [(x,y),(x+32,y+32),(x+32,y),(x,y+32)]

if tile_index == 5 angle = 45 and vertices = [(x+32,y+32),(x,y+32),(x+32,y)]

if tile_index in [6,7] angle = 22.5 if 6 : vertices = [(x+32,y+32),(x,y+32),(x+32,y+17)]

if 7 : vertices = [(x+32,y+32),(x+32,y),(x,y+32),(x,y+16)]

if tile_index in [8,9,10] angle = 11.25 if 8 : vertices = [(x+32,y+32),(x,y+32),(x+32,y+23)]

if 9 : vertices = [(x+32,y+32),(x+32,y+26),(x,y+32),(x,y+12)]

if 10 : vertices = [(x+32,y+32),(x+32,y),(x,y+32),(x,y+11)]

if tile_index in [11,12,13,14,15] if in [11,12] : angle = 63.4

if in [13,14,15] : angle = 72.6

if 11 : vertices = [(x+32,y+32),(x+32,y),(x+17,y+32)]

if 12 : vertices = [(x+32,y+32),(x+32,y),(x,y+32),(x+15,y)]

if 13 : vertices = [(x+32,y+32),(x+32,y),(x+23,y+32)]

if 14 : vertices = [(x+32,y+32),(x+32,y),(x+12,y+32),(x+22,y)]

if 15 : vertices = [(x+32,y+32),(x+32,y),(x+10,y),(x,y+32)]

if tile_index == 16 angle = 135 and vertices = [(x+32,y+32),(x,y),(x,y+32)]

if tile_index in [17,18] angle = 157.5 if 17 : vertices = [(x+32,y+32),(x,y+32),(x,y+17)]

if 18 : vertices = [(x+32,y+32),(x,y),(x,y+32),(x+32,y+16)]

if tile_index in [19,20,21] angle = 101.25

if 19 : vertices = [(x+32,y+32),(x,y+32),(x+32,y+23)]

if 20 : vertices = [(x+32,y+32),(x+32,y+12),(x,y+32),(x,y+22)]

if 21 : vertices = [(x+32,y+32),(x,y),(x,y+32),(x+32,y+11)]

if tile_index in [22,23,24,25,26] if in [22,23] : angle = 116.6

if in [24,25,26] : angle = 107.4

if 22 : vertices = [(x,y),(x,y+32),(x+15,y+32)]

if 23 : vertices = [(x+32,y+32),(x,y),(x,y+32),(x+17,y)]

if 24 : vertices = [(x,y+32),(x,y),(x+10,y+32)]

if 25 : vertices = [(x,y),(x,y+32),(x+11,y),(x+21,y+32)]

if 26 : vertices = [(x+32,y+32),(x,y),(x+10,y),(x+23,y)]

for the 4 last tiles, they were updated later

26/03 : We're finally seeing the light in the tunnel, while the physics is still wacky, it mostly works

03/04 : finally we can shoot without much problem

10/04 : frictions are finished, we just need to comment the code and tidy it up

14/04 : loading the image background, and putting the whole code in a function for a better merge

function used in debug : def draw_hitbox(ball, tile):

```
  if collision_check(tile.vertices, (ball.pos.x, ball.pos.y), ball.radius):
     pygame.draw.polygon(screen, "green", tile.vertices)
```

15/04 : Ice tiles can break

Major difficulties : -creating a realistic physic, with good frictions

-SAT algorithm

-vertices of the tiles were determined by counting manually the pixels

weakness : To ensure a smooth and seamless movement, 120 fps must be enabled. Otherwise, the ball may go through the walls

Taha : Power Up Log

10/02

Since I did not have the physics necessary at first to make my powerups accordingly,

I simply made a quite simple algorithm to make the ball go in an arc,

I then made my sticky powerup, but I struggled at first because the ball either was stuck before even launching or did not even stick. But after a while I finally made it stick when it landed.

3/03

A couple of weeks passed, I finally had the physics (thanks to Jalil) and had to remake the powerup from scratch)

because it did not work with the new code.

The main problem I encountered is that if the ball spawned directly with the sticky power, launching it became impossible because the ball was already stuck to the ground before even hitting it. I then decided to have a key to turn it off and on, so it did not cause any problem (press the A key).

17/03

Made a fast fall powerup (still in testing area still imperfect) press e to activate it and turn the ball blue. I just need to wait until I have the maps ready so I can modify it accordingly.

slightly tweaked the physics because the ball was too bouncy for a golf ball, so I changed it a bit.

imported and slightly tweaked the fast fall and sticky ball powerups to work with the new physics (collisions/tiles)

7/04

made the bouncy ball powerup but still need some work as the collisions are still a mess as of now.

(the ball goes through walls)(press z to use) also added a ball reset key bind (R) so when the ball just phases through the walls I can bring it back, it’s still far from done because I need to reset it depending on the spawn points but because I don't have spawns yet I just made it come back to the center

of the map

24/04

The bouncy ball was too buggy. The ball kept gaining speed forever and started bouncing through walls and noclipping all the tiles. fixed that by capping the max bounce speed to 1.90x(I think) the current speed (lower speeds were making the ball not feeling bouncy at all and higher Mult did not fix the problem).

Now it feels bouncy.

still might tweak the retention later depending on how it feels on different maps.

Frédéric : Level Design Log

03/02 : concept arts for the levels

26/02 : finished grass levels

17/03 : finished sand levels

24/03 : finished ice levels

05/05 : Flag class prototype

10/05 : Flag put in each level

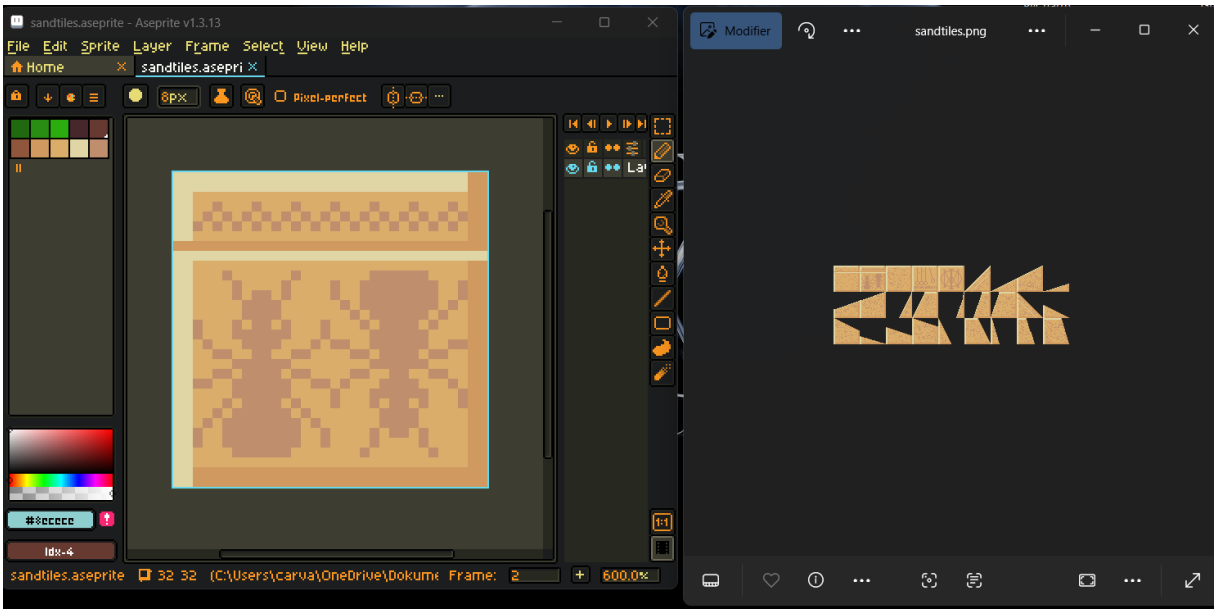
Mathias Carvalho

Mathias : Art Log

1st part : Sprites and Visuals

We settled down for a *Pixel Art* aesthetic as it was simple to implement and to create. The game's dimensions of 640x480 was chosen for both its simplicity of implementation and its 4:3, "retro" style feel.

I used the software *ASEPRITE* for the creation of the sprites, for its utility when it comes to automatically creating sprite sheets from drawings made in the software.

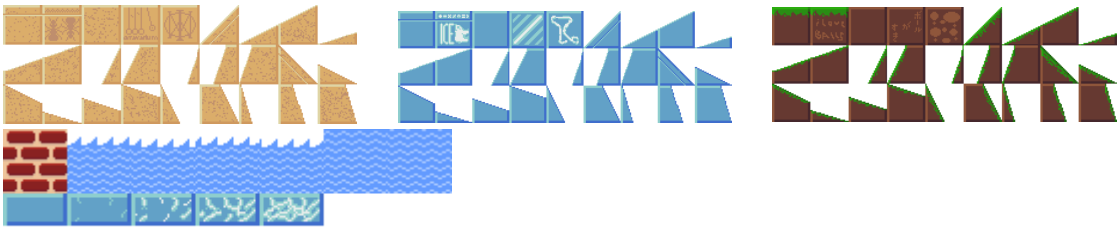


I started working on the game sprites on 03/02/2025, during the first class fitted out for the project. Sprites were noted as "completed" on 25/04/2025, although they might have been subject to small changes since then.

3 types of sprite sheets are predominant in both their utility and the amount of work put into them, they were made in multiple iterations to support the game's multiple "Areas" :

- The "Buttons" sheet, including all the buttons used in the level selection UI

- The three “Tiles” sheets (and the “Extra tiles Sheet”) used to make all different level Tilemaps (which were later combined into an “alltiles” files to avoid redundancy).



- The three “Background” sheets, for each of the areas’ level selection screen and every level background

The latter type of sheet indubitably was the toughest to make, as it needed rigor in its size (keeping the 640x480 size for each scene), but also creativity and thoughtfulness for each scene, to make each of the backgrounds both coherent in its area’s setting and recognizable to give the game its identity.

Thus, each area was given its proper identity and color palette (respectively “Natural Flora” and green for Balling Hills zone, “Egyptian Desert” and beige for Lines in the Sand zone, and “Icy Gnome Land” and blue for Nedellekshögg zone), which hopefully gave each zone its own feel. *Some slight references to my own liking were implemented in the last two zones.*

The choice of a slightly darkening blue filter on the level backgrounds was made to easily differentiate the foregrounds for it and not confusing the own level tile sheet with it.

Were also added in some sprites to better the interactivity and the information given to the player, such as a screen for when the level is beaten, names for each level, and a tutorial screen.

Starting April, I also implemented lots of sprites, such as the level end screen, or the level name at the top.

2nd part : Music and Sound effects

The music for the game was created by me, all by pure inspiration. I started working on the music at the beginning of April.

The music was made using the free LMMS software and taking instrument rips available for free on the internet.

The game includes 4 songs :

- A “Nature” sound for the title and world selection screen (found copyright-free on the internet)
- A chill music for the Grass zone
- An upbeat and mysterious music for the Sand zone
- A melancholic music for the Ice zone

All the music implementation in pygame was done by me on Thursday 8th.

Additionally, some ideas were given for implementation of sound effects (such as voice lines for level end appreciations or ball hit sound), but with the deadline approaching and the lack of professional equipment, these concepts didn’t make it through.

Victor : Menu Logic and User Interface Log

March

Getting a hang of pygame and the multiple limitations that I will encounter (and overcome) with the game engine, mainly with how the sprites work (i.e. how to load them, and how to make them show up on screen).

A big part of my work early on is to talk with Mathias, both to put some bonds over his creations, to ensure that I can later implement them, but also to work on some prototypes with placeholder images. The goal is also to figure out how the user will navigate the menus. We finally settled on a world and levels system that the player will navigate through buttons.

To make the whole thing more interactive and less static, we also decided that the buttons will change when they are hovered. Figuring out how to replace sprites was quite the challenge, when I didn't as much experience with pygame! I also had a little bit of fun implementing the splash text that changes size. The logic is quite elementary, but the result makes the title screen (the first impression of the game that a player has) feel so much more alive!

April

This month is when I felt like I progressed, my work was much more efficient, I could do in one sitting what I would have done in multiple days two weeks ago.

The main things implemented during this month were the button to go back in previous menus and the link between buttons and the actual game.

At the end of the month, I also completely rewritten my code for it to be more readable and more easily modified by the other groups members, because navigating it before was like walking through a maze.

May

This month was the one of various fixes and finishing touches. I applied some changes to the physics algorithm and to the gameplay, such as making the ball respawn when it touches water, with the help of Jalil and Taha. I made the levels use a global sprite sheet and I added some feedback when the player wins the game, both with the help of Mathias and Frederic.

It was great seeing the team come together for the final stretch.