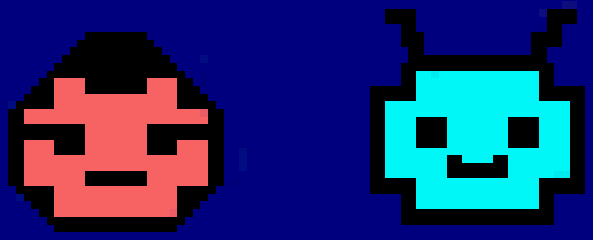
C19-Retro Team

Cell Block



Have fun!!

C19 Retro Team

Cell Block

Arismendi Sánchez, Carlos Eduardo[1]; Chernysh, Antón[2]; Cortés Espinosa, Sergio[3]

[1]carlos.arismendisanchez@gmail.com
[2]anton_chernysh@outlook.es
[3]sercotes93@gmail.com

November 3, 2020

Contents

1 Introduction and context

Welcome to Cell Block game. This is November 26th, 2020. The world has been devastated by a global virus named coronavirus. This virus has converted the living into zombies and the dead into ghosts. This problem began in Halloween 2020 because people, in case of not receiving sweets, sneezed at neighbours, resulting in a global pandemic.

Our character named Jimmy, is the last known survivor and he is in great danger. To get to safety he needs to get out of the forest that is full of enemies. It will not be an easy path, but we are sure that you can help him to achieve it.

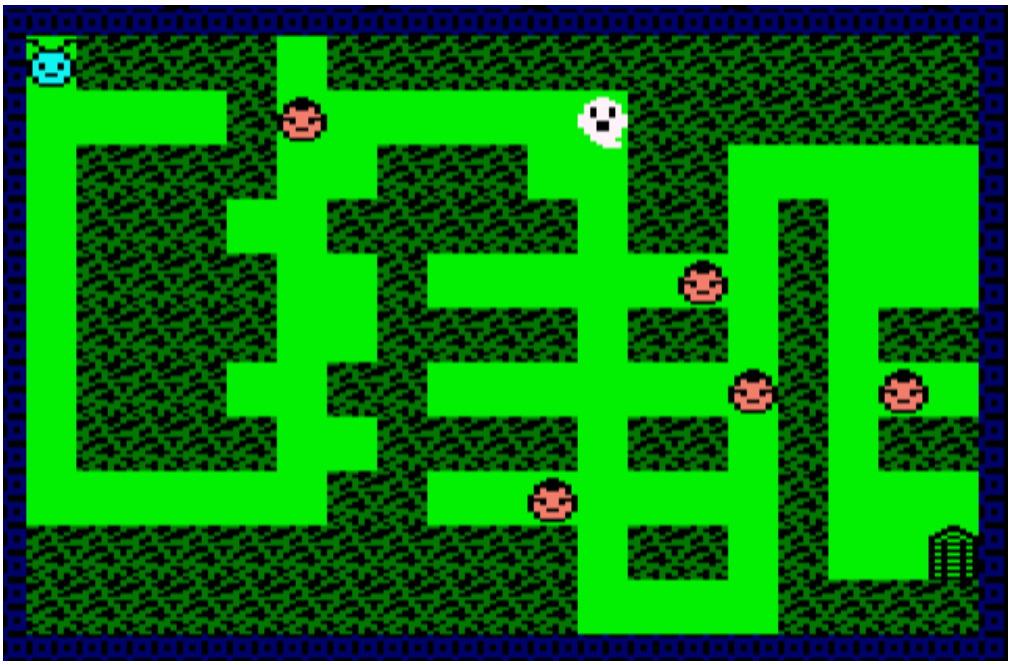
2 Load Instructions

2.1 Amstrad CPC 464

Inserts the cassette in the tape unit and check that the cassette is fully rewound. Then type *RUN*" and press ENTER. The load screen will be displayed and the game will start.

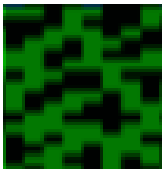
3 How to play

Your objective is to overcome all the levels without allowing zombies to infect you. To pass each level you must dodge the zombies that are on your way until you reach the exit of each map.

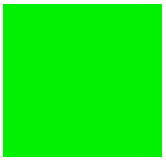


Each enemy controls an area of the map. You need to be stealthy and fast so they do not infect you. In more advanced levels you will meet a new enemy that has the ability to cross obstacles. You will need ingenuity to distract him, and be able to reach the exit.

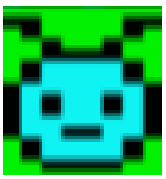
The levels are composed of the following elements, with which you can interact to achieve your goal:



SOLID BLOCK
Solid wall through which you can not move



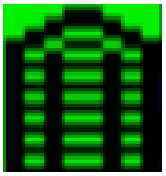
GRASS BLOCK
Grass block through which you can move



PLAYER
Jimmy, the last survivor

ZOMBIE ENEMY
Enemy patrolling an area of the map

GHOST ENEMY
Enemy capable of going through walls and chasing you



EXIT
Exit door of each map allowing to reach the next level

4 Controls

When the game is loaded you can see the main menu with the keys for movement:

Moreover, the player can use some extra keys to pause the game, restart the whole game or continue playing.

ESC: Pause game and open a menu with options.

SPACE: Start a new game or in case of death restart the level.

R: Restart the whole game, setting the lives to the maximum (3 lives) and starting from the first level.

5 Tips

- The maps contain shortcuts to roads with higher risk for experts who want to pass the level quickly. It will depend on you which way you want to take.
- Basic enemies do not chase you, if you get to a safe place, take your time to think about your following steps.
- All levels have been tested one by one and can be solved! When you find yourself stuck and can not stop dying in the same place, take your time to think.
- In levels with a ghost you do not have much time to think. You must be fast and think how to distract him allowing you to pass to the exit.

6 How it works?

It is important to mention that the video mode used for this game is mode 0 (160x200 pixels).

6.1 Load Screen

For the design of the load screen we used Gimp and we made a simple design, but that is not the important part, that one you can see just by opening the game. The important part is the problems we had implementing the loading screen and how we solved them.

When we first implemented the loading screen, we saw how the loading screen was perfect, but then, when the game started, all the colours in the game had been changed and it was a disaster. Then, we figured out that implementing the loading screen changes the colours palette of the game, even if it is the default command for the loading screen, and, since we had our palette in the image conversion, we had to initialize the palette at the start of the game. Or that is what I would want that to say, actually the truth is that we discovered that right after we changed all the colours of our game, all the sprites and we really lost a lot of time. I hope that you, by reading this, do not make the same mistake.

6.2 Music

For the creation of the music, we used Arkos Tracker 1.0, the version CPCtelera macros work with. We have two different songs and they are both quite simple, since we do not know much about music or composing music. We made a song for the in-game and another one for the menus, but then, trying to implement them, we met a nice bug that created ”another song”, it was indeed one of the two songs, the one for the menus, but at a quite slower speed, making it seem like a third song, and we actually liked it and decided not to fix that bug. We finally put the song for the in-game in the intro and the menu song for the menus, having the extra-song for the in-game. Maybe it is not the best of the results, but according to us it is quite good according to our knowledge of music and how Arkos Tracker works.

6.3 Managers

6.3.1 Entity

This manager is the one that manages the initialization and storage of entities enemies and player. All entities has the same fields although not all entities use all fields. For example, player entity does not use the counters nor increments of velocity nor flags:

1. **Screen coordinates (x, y)**.
2. **Map coordinates (cx, cy)**: The maps are divided into 19 width blocks and 11 height blocks, each one of 4 bytes (8 pixels) width and 16 bytes (16 pixels) height. The purpose of this block size is to make it look like a square.
3. **Velocity (vx, vy)**: This indicates how many bytes the entity move and the direction. Only the player is allowed to move in diagonal. Besides, only one block per movement is allowed (e.g. right movement, is 4 bytes to the right).

4. **Velocity counters (counterx, countery)**: the movement velocity of enemies is controlled by these counters in order to allow slower velocities. Each time the *counterx* or *countery* reaches a predefined value (100), a flag in the entity allowing it to move is activated.

5. **Velocity increments (incrementx, incrementy)**: the previous explained counters are incremented by this increments, which are specified at the initialization of the entities. The higher these increments are, the faster the entity moves.

6. **Previous memory video pointer**: this points to the position of video memory where the sprite of the entity is drawn in order to remove it in next iterations of the program.

7. **Lives**: indicates how many lives the player has. At the beginning of the game the number is 3.

8. **Map index**: the map is stored as an array of bytes (1 bytes per block), so this index allows to access directly to the map block where the entity is positioned in order to check the collision between the entity and the map faster.

9. **Flags**:

    - Movement: indicates whether the entity enemy can move or not. This is used both to know if it is necessary to check the entity collisions and to know if it is necessary to render the entity in a new position.
    - Ghost: indicates whether the enemy is a ghost or a normal enemy. This is used to know what collision and render functions apply to the entity since there are different checks and functions depending on the enemy type.

6.3.2 Map

The map manager is responsible of managing the levels of the game by giving functions to other components to get the level number and a pointer to the present map, and an initialization function to set the map to be the first one (level 1).

The maps are stored as arrays of bytes where each byte indicated the block type (wall, grass or exit).

6.3.3 Game

This is the manager where all the other managers and systems converge. This is main component and it takes care of managing the game by initializing the game calling the necessary managers and systems and also updating the game by calling the input, IA, physics and render system.

6.4 Systems

6.4.1 Input

This just checks if the player has pressed any key, which key and executing the proper action depending on the specific key (e.g. ESC key calls the game manager to pause the game and once the game is resumed, renders the map again.

When this system detects that the player pressed a movement key, depending on which key (O, P, Q, A), the player entity velocities are updated with predefined values to allow the movement of the entity to a given direction (e.g. P key, sets the velocity on X axis to 4, which allows a movement of 1 block to the right).

6.4.2 Physics

- **Enemies**: the system uses the *incrementx* and *incrementy* fields of the entities to update the counters and once *counterx* or *countery* reaches 100, the entity movement flag is set to active allowing the entity to move one block in the corresponding direction indicated by its velocity.

  Besides, before "confirming" the movement, it calls the collision system to check if the entity collides with a map wall, in which case, the velocity is changed in sign so the entity now moves to the opposite direction.

  If the movement is canceled, the flag of movement is deactivated.

- **Player**: physics system updates the player position taking into account the velocities given by input system. Moreover, physics also checks the collision with the maps blocks as it does with the enemies.

  This system also checks if the player collides with an enemy by comparing their maps coordinates or if the player collides with the ghost by comparing their screen coordinates. In both cases, if the collision occurs, the player is subtracted a life.

6.4.3 Render

It takes care of drawing the different maps, entities and menus (these by calling the menu system).

- **Maps** are drawn at the beginning of each level and after closing a menu (e.g. after closing the pause menu).
- **Player and enemies** are drawn each iteration only if they have moved in relation to the previous game loop iteration.
- **Menus** are drawn by calling the corresponding function of the menu system.

6.4.4 Menu

It provides a series of functions for drawing different menus such as initial game menu, pause menu, die menu and end of game menu (the one displayed when the player achieves the exit of the last level).

6.4.5 Collision

The collision system is divided into three parts:

- **Collision enemy-player**: the positions of the enemy cell are compared with the player, and returning if there is a collision.

- **Collision entity-map**: using the entity cell the system calculates corresponding pointer to map-cell and compares if it is solid.
- **Collision ghost-player**: in order to check the collision between player and ghost, the coordinates of both entities are projected into their corresponding axis (e.g. the X coordinates into the X axis and the Y coordinate into the Y axis). For both entities, we have two X coordinates (the beginning of the entity X and the end of the entity $X + Entity_{width}$). Consider x_1 the X coordinate of player and x_2 the X coordinate of ghost. Once the projections are made, there are 4 options:

  1. x_1 is greater or equal than $x_2 + E_w$: so there is not collision.
  2. x_2 is greater or equal than $x_1 + E_w$: so there is not collision.
  3. x_1 is greater than x_2 AND x_1 is less than $x_2 + E_w$: so there is collision.
  4. x_2 is greater than x_1 AND x_2 is less than $x_1 + E_w$: so there is collision.

  Same reasoning applies to Y axis. In case the entities collide in both axis, a collision takes place in the game. Otherwise, for example, only collide in one axis but not the other one, no collision occurs.

6.4.6 AI

The only enemy with AI is the ghost. His mission is to chase the player until he collides with him. In each game cycle, the AI calculates the speeds of the ghost allowing him to move towards the player.

7 Prince of Persia

To transmit a nod to the game Prince of Persia we have created a level that contains the word of Persia and also, it is the first map where the player meets the ghost.

8 Credits

Code & Graphics: *Anton Chernysh & Carlos Eduardo Arismendi Sánchez*
Loading screen & Music: *Sergio Cortés*

And our sincere thanks to:

- Fran Gallego, for creating CPCtelera and for teaching us not only to program but to debug and really understand how a computer works.
- Our class group for always helping each other