Game Development Report
The Last Brew (TLB)

Eric Ayllón Palazón, Vicent Baeza Esteve, and Clara Gonzalez Sanchez

Universidad de Alicante, España
eap56,vbe5,cgsg4@alu.ua.es

1 Introduction

This document gives some insight on how **The Last Brew** was developed. Each of the following sections will talk about each stage of the development starting from the warm-up that is not closely related to the actual game but still provided some of the knowledge and basic structure used for the development of the game in Z80 assembly language. During stages described in sections 2, 3 and 4 the team was guided by some provided resources which include materials given to them by their university teacher and a couple of sets of streams of their university teacher.

2 First contact with the Z80: video memory and machine language

The first steps on the development process were focused on learning how the Z80 and its memory layout work, specially the video memory. All the members of the team learnt how to manually draw sprites by writing the appropriate values into the ram of a CPC 464 emulator called WinApe. All the subsequent development was tested mostly on this emulator, although during some of the final versions were also tested on a physical CPC 464.

This first contact provided some knowledge on how some of the instructions were encoded in machine language for the Z80. Nearing the end of this period and the start of the first actual development stage (section 3) the team was provided with an assembler and a linker for the development of software for the Z80 in assembly language instead of having to manually write instruction codes and data directly into the ram.

3 First Development Stage: Starfield effect

During the first stage, the team developed a star field effect in assembly language following the same principles of the ECS architecture used in the first two videos of this playlist. Note that the videos explain how to do the effect in C, but the development team did it in Assembly language.

This stage gave the team some insight on how the ECS architecture works and it was crucial since all further development followed the general idea of this architecture. This provided a tool for managing entities, updating them and rendering them. The render system during this stage was not an advanced rendering system and it was not suited for drawing sprites.

Some knowledge on the cpctelera library used as a development supporting library was also acquired during this stage.

3.1 Technical details

The subsystems added to our tools during this stage are:

- An **entity manager** in charge of entity definition and the following operations on entities:
  - Creation
  - Iteration
  - Destruction
- A **physics system** in charge of updating the physical aspects of the entities.
- A **render system** in charge of drawing the stars based on their position and color.

The most important detail of this stage is the entity handling, the entities are stored in an array-like section of the memory. A pointer to the start of said section (called *entities* from now on) and a pointer to the address after the last valid entity (referred to *nextFree* from now on) are kept at all times in order to manage the iteration and the rest of the operations. The deletion of entities follow the *copy-last* strategy in which a deleted entity becomes a clone of the last entity and updating *nextFree* in order to invalidate the last entity so that it is not kept as a duplicate.

3.2 Organization

The team organization during this stage followed an independent development scheme where each member of the team developed their own version of the starfield effect. After this stage, the team randomly decided on one of the three versions to keep as the starting point for the next stage since all three versions were equivalent.

4 Second Development Stage: Atari Assault

During the second stage in the development process, the team recycled the code from the starfield effect to have a solid starting point for developing the skeleton of an Atari Assault clone. The objective of this stage was to have the bare bones of a simple and generic game engine ready to be extended and implement an actual game.

This stage became a bit more challenging than the previous one since the idea was to spend at most the same amount of time as in the previous stage. Just like in the first stage, the team followed the direction given by the last three streams of this playlist. Again, the streams explain the process in C, but our team continued the development in Assembly language for the Z80.

The knowledge of our team on the ECS architecture was expanded and consolidated a bit more thanks to this stage. The render system was refined a bit more and a couple of systems and managers were added (see section 4.1).

4.1 Technical details

The subsystems added to the project during this stage are the following:

- A **game manager** in charge of:
  - initialization of the game
  - definition of a proper game loop
- An **animation manager** which defined the structure of animation structures
- An **animation system** in charge of updating the animations and sprites of animated entities
- An *AI system* which defined different entity behaviors and applied them every frame
- A **collision system** which checked for collisions

There are some important details about the subsystems. First, the render system was now suited for rendering sprites therefore entities now had a pointer to their sprite. Next, the physics system now also checked for input for user-controlled entities.

For the AI system, each entity had a pointer to their behavior or a null pointer (value of 0) if no behavior was employed.

For collisions, a special byte was reserved for the definition of what entity types can an entity collide with.

4.2 Organization

The organization of our team during this stage was to have only one member working on the project at the same time since we would watch the same streams and development of later parts required having the previous ones done, there was no use on having two members working at the same time.

5 Third Development Stage: cloning GOOMS mechanics

The third stage of development was cloning the base mechanics of the game GOOMS, presented on CPCRetrodev 2021. The idea was to adapt our generic game engine to have a game engine capable of doing more or less the same as in GOOMS.

For reference, we had to download the game and play it, we also watched their video showcase since that way we could also pause and pay attention to as many details as possible.

Since the idea was **not** to just clone the game, we skipped the sprite drawing and worked with the sprites we had from the previous stage or just color boxes as placeholders for sprites of specific size we needed.

During this stage we discussed all the basic mechanics we had to replicate. Some of them were: having a target to defend, a score system, different enemy types, a level system, shooting and moving in different directions. A complete list is provided in section 5.2 along with an indicative of who was in charge of each item.

5.1 Technical details

The additions and changes introduced to the project during this stage are listed below:

- Changes on the physics system
- An input system
- Changes on the collision system
- Knock-back on collision
- Two enemy types with their corresponding AI.
- A score system that would be updated every time an enemy died.
- A game interface indicating the health of the player and that of the defense target.
- A level manager

The physics system was changed in order to allow vertical and horizontal movement. Furthermore, since the shooting would be in any vertical or horizontal direction, each entity would have a byte of flags indicating their physical interaction like the direction they are facing, if they move during the current frame and if they shoot during the current frame. Depending on the value of this flags, the entity moved in the appropriate direction and created a bullet in that direction. Shooting is tied to the direction the entity is facing, this makes the bullet initialization easier since its directions would be the same as the parent entity but removing the shooting flag. This also removes the need for negative values in speed, although it complicates a bit more the movement logic.

The input checks were segregated from the physics system into a separate system, the input system. This system checks the input controls for all entities that can receive input. This could allow for mechanics like transferring the control from the player to another entity, like for example the defense target, or shooting a bullet that gave the player control over the hit entity. Although this mechanics have not been implemented in the game. This system was responsible of setting the right flags on controllable entities for the physics system to make them interact as they should.

The collision system was adapted so that custom behaviors were executed for each type of entity depending on what other entity it collided with. This allowed for the enemies to die instantly when colliding with the player but only taking one point of damage from shots.

The new enemies had an AI that targeted the player or the defense target and followed them. The enemy would first try to get to the same column as the followed entity and then try to reach the same row. The AI system was responsible of setting the same flags the input system modified for the controllable entities but the AI system modified them for the entities with AI.

For the implementation of the score system, since the score system would have to be converted to ascii in order to print it, the score was stored as 4 bytes, one for each digit in the score, and then the maths on the score would implement a quick modulo operation presented in pseudocode in listing 1. This implementation makes use of the assumption that the number is strictly lower than twice the value of the modulo base, this true since the score of any enemy is 1, in the worst case scenario, score digit is 9, 1 is added and maybe another 1 for the carry. The restrictions are always met. Another detail about the score system is that since the frequency of updating the score was much lower than the frequency of display, the score values were stored as the ascii characters to be printed and converted back and forth to numbers when the score needed to be updated.

Algorithm 1 QuickModulo(N, M)

```
Require: $0 < N < 2M$
  $N \leftarrow N - M$
  if $N < 0$ then                          ▷ If N was lower than M
      $N \leftarrow N + M$                     ▷ Restore its value
  end if
```

The game interface printed each frame the health of the player, the health of the defense target and the current score. In order to clear the lost health points, a black square is drawn after the last point of health both for the player and the target. This would cause visual bugs if the player or the target take two points of damage on the same frame but the chances were low enough to leave that detail for a later stage.

The level manager defined the level structure as a pointer to some piece of text to be written at the start of the level, a displacement indicating where the next level starts and then a series of pointers to the templates of the enemies that spawn during the level. The level manager also managed the menus and ending scenes.

The level manager also managed the enemy spawning by using two pointers: nextEnemy and nextLevel, as well as a delay counter and an enemy counter. Every frame the manager checks if nextEnemy and nextLevel have the same value, this means there are no more enemies to spawn for the current level. If there are still enemies to spawn the delay counter decreases, if it reaches 0, then an enemy spawns, the enemy count increases and the delay counter is reset. If there are no more enemies to spawn, the enemy counter is checked and nothing is done if its value is not 0, if its value is 0, then the next level starts. Every time an enemy died, the enemy counter would be decremented.

5.2 Organization

During this stage, our team decided to divide the work in the following way:

Eric	Changes on the physics system
Eric	An input system
Vicent	Changes on the collision system
Vicent	Knock-back on collision
Vicent	Two enemy types with their corresponding AI
Eric	A score system that would be updated every time an enemy died
Eric	A game interface indicating the health of the player and that of the defense target
Eric	A level manager
Clara	Visual design

During this stage we decided that although the sprites and animations were none of our priorities, we had most of the development under control and assigned Clara the design of the sprites, animations and the til-emap so that she had enough time to make them look as good as possible and then have time in later stages for their integration.

6 Fourth Development Stage: developing own content and mechanics

In the fourth stage of development we considered the game had a solid base to introduce some changes and start refining some rough edges.

From this stage to the end, we had no model to copy, however, we considered interesting to take into account some examples of other games in terms of technical decision, or at least what we believed other game developers decided. This examples only served for deciding if investing more time on some things was a waste of time or not.

We decided the topic of the game and we wanted to have a reference to the Super Cauldron game, therefore the objective of the game would be to defend Zmira (the main character of Super Cauldron) while she prepares a potion.

6.1 Technical details

The additions and changes introduced to the project during this stage are listed below:

- Changes on the animation system
- Changed the Atari Assault for the new sprites
- Integration of the tile-map

– The AI of a shooter enemy
– Enhancement of the AI system
– Changes in the input system
– Changes in the physics system
– Proper ending and game restart

The animation system was rewritten to support directional animations, this means, entities would have different animations depending on the direction they were facing.

The sprites and tile-map were integrated in the game, therefore we stopped using the Atari Assault sprites and color boxes. The tile-map was later moved to be drawn only when a level starts in order to not waste cpu time since the entities should never touch the borders and their sprites had a border of the same color as the floor so that they redrew the floor upon moving.

The new enemy had the ability to shoot the player, therefore it only moved vertically or horizontally to find the player and then shoot when in sight. In order to balance this enemy, it had a cooldown for shooting and during that cooldown, it could not move at all.

The AI system was enhanced by introducing a stochastic update to enemies. A bit-mask on each entity was employed for this, along with a frame counter. Upon spawn, the enemies would take the value of the enemy counter and keep it as the update condition. Each frame, the AI system would check if the frame counter equals the update condition of each entity and only those for whom the value is true would be updated i.e. move or interact during that frame. The purpose of the bit-mask is to limit the amount of bits to be checked on this comparison. Equation 1 shows an example of three enemies and a frame counter. Enemy 1 updates on the given frame but enemies 2 and 3 do not. Enemies 2 and 3 spawned when the enemy counter was the same, this could have happened if an enemy spawns, then a previous enemy dies before the next spawn. Enemies 2 and 3 do not always update at the same time, enemy 3 can update once every 8 frames but enemy 2 updates once every 4 frames. This is because of the bit-mask. Since the bit-mask of the enemy 3 uses 3 bits, 2^3 values are possible, then once every 2^3 frames, the frame counter would have the value that allows enemy 3 to update.

$$\begin{array}{lcc} & \text{Update Condition} & \text{Bitmask} \\ \text{Enemy 1} & 000101{\color{red}11} & 00000011 \\ \text{Enemy 2} & 000110{\color{red}10} & 00000011 \\ \text{Enemy 3} & 000110{\color{red}10} & 00000111 \\ \text{Frame Counter:} & 010110{\color{red}11} & \end{array} \quad (1)$$

The input system was changed to support joystick controls which was easy to add since the joystick emits the signal of yet another key for each direction and button that was checked by the input system.

The physics system was changed to check if an entity would go off-limits and if so, set its position to the closest valid position. This was done by performing a min or max operation between the vertical position and the limits of the map and the same for the horizontal position.

Unlike in the previous stage, during this stage the level system was adapted to allow the player to start again after finishing the game either winning or losing. The losing state was implemented during this stage as well, after decreasing the health of the player or the target, if health reached 0, then the appropriate ending scene would be shown. This was implemented as two separate levels. In order to force the level change during the same frame, a routine sets the nextLevel and nextEnemy pointers both to the right level (playerDied or targetDied), then sets the enemy count to 0 in order to force the level change. Since the collision might kill an enemy the enemy counter now had to be checked for negative values (if it is negative after decreasing it, just increase it again) in order to prevent the level manager from freezing the whole game.

6.2 Organization

The tasks performed during this stage were distributed as follows:

Clara	Changes on the animation system
Clara	Changed the Atari Assault for the new sprites
Clara	Integration of the tile-map
Vicent	The AI of a shooter enemy
Vicent	Enhancement of the AI system
Eric	Changes in the input system
Vicent	Changes in the physics system
Eric	Proper ending and game restart

7 Refinement and Report

During the last stage of development we focused on just refining the game and writing the documentation. Our objective was to have the game ready for launch by October 30th although our actual deadline was November 3rd.

7.1 Technical details

The last additions and changes to the project were the following:

- Audio design
- Level design
- Changing the text color and font
- Change in game interface

The audio for the game was designed and created using **Arkos Tracker 1.0**. This tool was provided within the cpctelera toolkit.

The idea behind the new levels was to shorten as much as possible the text in the middle of the game and provide some variety on the enemy types and quantities. At least one level to introduce each enemy type was ensured and then in order to progressively scale the difficulty, the amount of enemy types that appeared on each level increased only when all possible enemy type combinations of the same amount of enemy types were used. This gave the following setups:

- 1 enemy:
  - Enemy 1
  - Enemy 2
  - Enemy 3
- 2 enemies:
  - Enemy 1 + Enemy 2
  - Enemy 1 + Enemy 3
  - Enemy 2 + Enemy 3
- 3 enemies: Enemy 1 + Enemy 2 + Enemy 3

The text color was changed to a color that allowed the player to read the content properly with a suitable contrast. We tried to use the same font as the one used on Super Cauldron font as another reference, but it did not work out as expected and had to use another font.

The game interface was redesigned so that the health and score are not drawn every frame, just when needed. In fact, health and target health are only drawn when levels start, along with the score and then upon taking any damage, the corresponding health point is cleared by drawing a black box over it. The score is now only redrawn when an enemy dies, just after updating the score value.

7.2 Organization

Vicent	Audio design
Eric	Level design
Clara	Changing the text color and font
Eric	Redacting the user manual and the Development Report (this document)
Eric	Change in game interface
Clara	Preparing the web page on itch.io

8 Planning and retrospective analysis

The time distribution for each of the development stages was inspired by the planning provided by the university teacher. Figure 1 shows what stage our team was working at for each day of development.

The time distribution was not an item we spent much time on, we tried to stick to the general schedule provided by the teacher. Furthermore, the general organization we had going was the following:

Eric	General project management, input, levels, game display
Vicent	AI, Physics, Audio design and implementation
Clara	Animations, Sprite, Pixel Art, Helped with Audio design

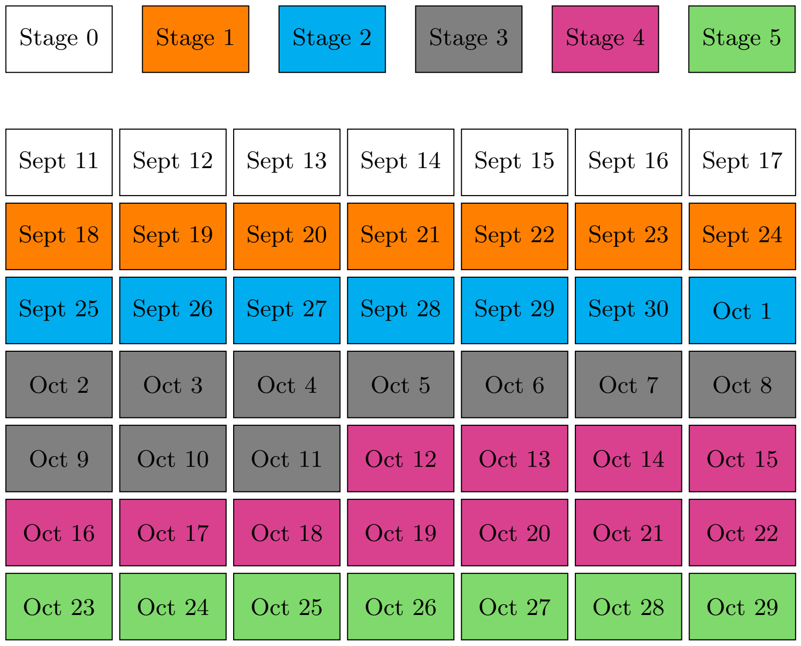


Fig. 1: Distribution of time among the development stages.

Finally, if I am going to give my personal opinion about our expectations and how we actually managed to develop this project, I believe we met the expectations about the time we took for each stage of development. Even though we did not have specific time frames defined for each task, we had everything relatively clear enough to proceed efficiently during the implementation of most of the components of the project. If we had encountered any serious adversity during development, we may have not finished with much leeway, but fortunately, that was not the case.

9 Personal conclusions

Eric After almost two months of development in Assembly for the Z80, I think I have learnt a series of things that I will always try to keep in mind in the future.

First, I can say is that even though I have missed being able to multiply two numbers, there usually is another way to design which is simpler and does not involve complex math.

Another thing I learnt thanks to this experience is that high level languages might save time by managing the memory keeping track of what programmers can and what they can not do, knowing at low level how the developed system works provides much more insight and allows the developer to make sure things work exactly as they believe things should work.

Vicent At the start, it seemed impossible to program a game in assembly in such a short time-span. This project has been quite the adventure, in which we have had to fundamentally rethink many things that we took for granted when programming. Having to keep track of registers, memory usage and execution cycles made it quite challenging to structure the project.

However, bit by bit, day by day, we learned and we improved. Two months ago, we could barely program in assembly. Now, we have made a whole game, playable from start to finish.

Clara After two months of dedicated effort, we've successfully crafted a fully functional video game in Z80 assembly. This journey has been an incredible learning experience, pushing our programming skills to new heights. We've encountered numerous challenges along the way, from managing registers to fine-tuning execution cycles. Nevertheless, through persistence, teamwork, and a commitment to improvement, we've achieved our goal. This project has not only enabled us to create an engaging game but has also deepened our understanding of low-level programming and the intricacies of the Z80 architecture. It stands as a testament to what can be accomplished with determination and passion. We take pride in our achievements, and this experience will undoubtedly influence our future endeavors in the world of programming

In the process of creating sprites for a video game, a fundamental step involves establishing a foundational character template, which then serves as the basis for generating variations to accommodate the design ideas for both the main character and enemies. This base character sprite is typically designed with the intent of reusing and adapting it to various in-game characters, thus optimizing efficiency in the sprite creation process. Figure 2 shows an example of this.

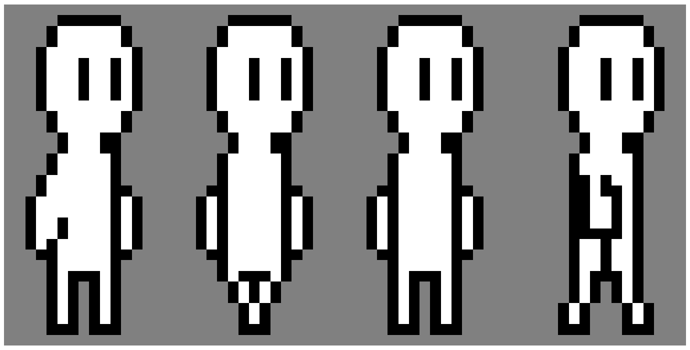
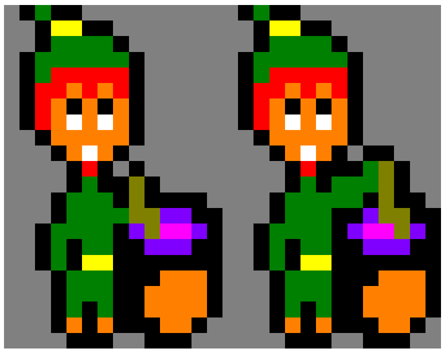


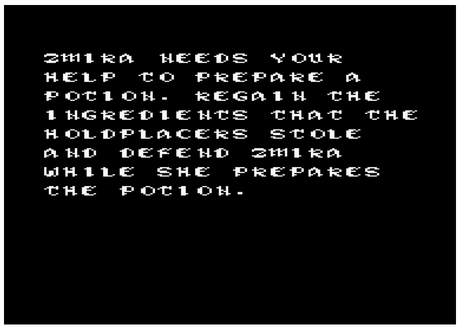
Fig. 2: Base template of the character walking right.

Regarding the animation system, each in-game entity has a field with a reference to an array containing the different animations it can perform, such as idle, walking right, walking left, moving up, and moving down. Consequently, the animation system operates by checking the current animation state and direction of the character. Depending on these factors, the system dynamically updates the appropriate sprite for the entity, ensuring that characters seamlessly transition between different animations to create a visually engaging and immersive gaming experience. This approach not only simplifies sprite management but also enables a versatile and responsive animation system that can adapt to the ever-changing demands of the game.

10 Easter eggs

Our game includes two easter eggs. The first one is a reference to the game Super Cauldron. The easter egg is that the character you have to defend is the main character of that game. Since it was not easy to make it look exactly like the original, we said that the defense target is Zmira in the text that tells the story of the game (see figure 3).


(a) Sprites of Zmira.


```
ZMIRA NEEDS YOUR
HELP TO PREPARE A
POTION. REGAIN THE
INGREDIENTS THAT THE
NOLDPLACERS STOLE
AND DEFEND ZMIRA
WHILE SHE PREPARES
THE POTION.
```

(b) Text mentioning Zmira.

Fig. 3: References to Super Cauldron.

The second easter egg is a commercial goodbye to the Amstrad CPC since they stopped being commercialized 30 years ago. Figure 4 shows this farewell.

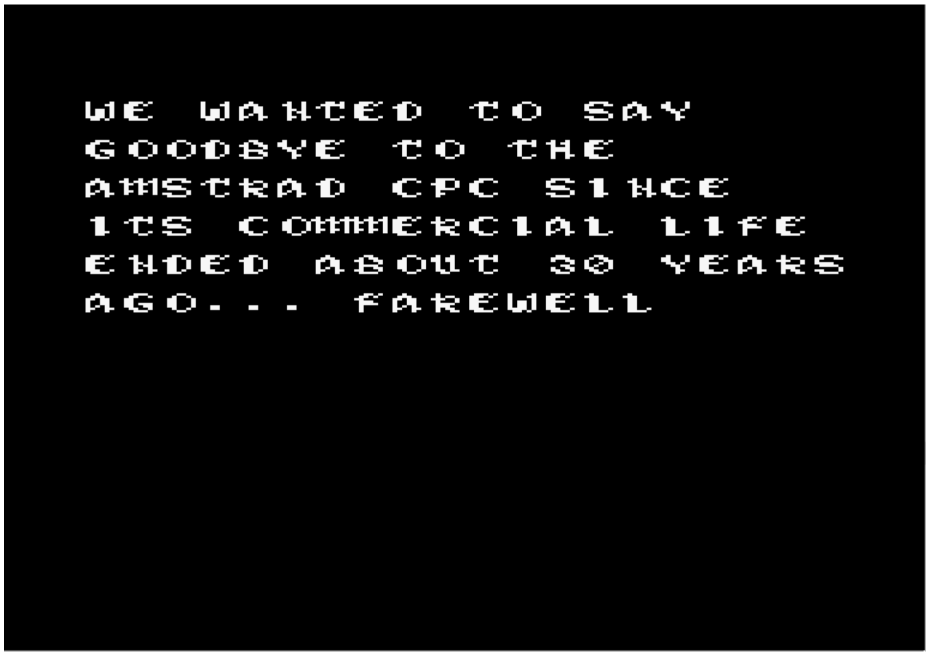

WE WANTED TO SAY
GOODBYE TO THE
AMSTRAD CPC SINCE
ITS COMMERCIAL LIFE
ENDED ABOUT 30 YEARS
AGO... FAREWELL

Fig. 4: Commercial goodbye for the Amstrad CPC.