A Framework for Metroidvania Games

Bruno Pinheiro Oliveira
Federal University of Ceará
Fortaleza, Brazil
brunopo9896@gmail.com

Artur de Oliveira da Rocha Franco
Federal University of Ceará
Fortaleza, Brazil
arturoliveira@virtual.ufc.br

José Wellington Franco da Silva
Federal University of Ceará
Fortaleza, Brazil
wellington@crateus.ufc.br

Fernando Antônio de Carvalho Gomes
Federal University of Ceará
Fortaleza, Brazil
carvalho@ufc.br

José Gilvan Rodrigues Maia
Federal University of Ceará
Fortaleza, Brazil
gilvanmaia@virtual.ufc.br

***Abstract*—Metroidvanias feature extensive maps that require proper exploration by reporting to skills acquired through the game, in addition to frequent fights against enemy waves. This gameplay style develops an enthralling experience built around factors such as curiosity and challenge, besides the possibility of evolving, obtaining rare items, finding unusual usage to well-known abilities, and unprecedented encounters. Metroidvanias are therefore fairly complex pieces of software that demand a considerable development effort. In this paper, we propose a framework for building Metroidvania games comprising the following key aspects of this genre: large map navigation; battle system; inventory; pathfinding; assistance system, allowing for getting help from companion NPCs (Non-Player Characters); skill tree; puzzle-solving; dynamic map loading; and recycling of NPC instances. A prototype game was built using the proposed framework to validate our approach. A specialized character tagging mechanic was introduced in our prototype with the view to demonstrate our approach is flexible enough to adapt to different mechanics.**

***Keywords*—metroidvania, game development, genre-specific framework**

I. Introduction

According to Silva et al. [1], games are complex software developed under the light of multidisciplinary areas. Among these, we can highlight artistic and communication-related areas like game design, visual design, sound design, and storytelling [2]. Moreover, there is plenty room for applying background knowledge from Computer Science, especially Computer Graphics (CG) and Artificial Intelligence (AI), among others involved in game development [3].

From a business perspective, the gaming market holds a prominent position in creative industry: its revenues even exceeds those from movie market, as it continues to grow at "Chinese" rates despite recent years of global market crisis. At the time this paper is written, Newzoo's market analysis[1] projects a growth of about 9.6% for 2019. Metroidvanias and similar platformer games, in their turn, are experiencing an ever-growing interest from the player community. For example, Celeste, a title that features many common qualities of a Metroidvania, was nominated in 3 categories for GDC Choice Awards 2018[2][3], including the main category *Game of the Year*. This is a major achievement for an indie game, considering the heavyweight titles also nominated, such as God of War and Red Dead Redemption 2.

A. Motivation

In industry, game development demands reducing production costs by resorting to modern methodologies, techniques, libraries, frameworks, and software tools specialized for game productions [4] [5] [3]. This paper proposes a specific game engine architecture for Metroidvania games. Metroidvanias are referenced by an amalgamation coined after two well-known franchises, Metroid and Castlevania, and comprise a sub-genre of popular action and adventure games. A typical Metroidvania features big level maps with non-linear exploration gameplay. Such maps contain a myriad of items and enemies of different levels of difficulty [6]. Specific features found in Metroivanias are discussed later in this paper.Some academic papers aim to explore other possible problems such as automated content generation [7]. Some techniques such as evolutionary algorithms were employed by Rodríguez, Cotta and Leiva [8]. That being said, the proposed framework is specially designed to support common *gameplay* elements found in Metroidvanias titles. Moreover, our framework also presents an effective design for the sake of convenient reuse for correlated game projects. We also developed a working prototype game in order to evaluate our framework in a real scenario.

Our framework also considers performance constraints we advocate are valid for both indie and mobile platforms. So, our game prototype is suitable for execution on affordable, low-cost machines with low processing power by resorting to optimization strategies in order to provide games with fluid gameplay and execution.

For example, pathfinding [9] is a complex problem to be circumvented in the aforementioned game genre, otherwise maps would not produce combat gameplay. However, reusing existing solutions typically implies complex reworking because of

[1]https://newzoo.com/insights/articles/the-global-games-market-will-generate-152-1-billion-in-2019-as-the-u-s-overtakes-china-as-the-biggest-market/

[2]https://twitter.com/celeste_game/status/1081313698000588806
[3]https://t.co/48l2OSSbt1

the peculiarities of this game genre [10]. For example, in a typical Metroidvania, there is a lack of total knowledge about the scenario, which is usually very broad to fit in memory and consequently must be loaded dynamically as the player steers her avatar [11]. Typically resource-constrained platforms such as game console demand map paging or procedural generation, or even a combination of those two methods [11] [3]. In light of what has been discussed in this paragraph, it is valid to assume that reusing existing pathfinding solutions into a Metroidvania game project demands, at least, integration efforts.

In addition, NPCs found in most titles usually exhibit deterministic, predictable, and boring behavior because they must go through a predetermined trajectory [10] [6]. Such limited approaches usually prevent NPCs from surprising or challenging the users [11]. On the other hand, existing steering behavior algorithms for NPCs may also display visually unpleasant results [11] [6].

B. Contributions

As far as we know, there is no similar framework in literature (see Section II-D). These are the main contributions of this paper:

- Game Map Management. Avoids memory overload and enhances reuse of common elements.
- Character recycling. Improves performance during saving, loading, restarting game zones, creating enemy waves, and other common operations during gameplay.
- Pathfinding and basic path planning [12], which abstracts the intricacies of integration with the Game Map Manager. Moreover, this component also adapts methods found in the literature, usually developed for 2D or 3D case, to work in the case of side-scrollers.
- Combat System. Designed to prevent game-breaking behavior in terms of infinite combos [13]. In particular, we compute damage based on data assigned to animation frames during attacks, thus allowing for inflicting varying damage over time. Moreover, the combat system is integrated into an ability tree, so our results can generalize to virtually any moves made available in combat.
- NPC Behavior System. We implement a combination of outstanding characteristics found in previous models in order to produce an adequate AI for the game genre [2] .
- We implemented the proposed framework on top of Unity 3D.
- We present practical evaluation results by means of a prototype game built on top of our framework. Our prototype features mechanics such as character tagging, skill tree, player following, and a customized combat system. Results were evaluated by users in a focus group.

C. Paper Outline

Our work is hereby structured in six sections. First, we address related work and theoretical background in Section II. The proposed framework is detailed in Section III and its implementation aspects are discussed throughout Section IV. Section V, in its turn, is devoted to experimental evaluation and results. The concluding remarks about our study are shown in Section VI.

II. Related Work

This section is divided in order to address the specific topics related to study, the definition of the main related concepts and a general survey of the requirements of a Metroidvania game.

A. Electronic Games

Schuytema [4] defines games as playful activities performed by means of actions and decisions which lead to a final state [10]. The act of play is limited by a set of rules governing an usually bounded universe. In digital game context, these rules and the many representations of the game state are controlled by an electronic machine, the computer [4]. Moreover, rules govern what are the consequences of actions and decisions taken by players. Besides, rules can be seen as a central design tool, which can provide challenge by making it difficult or fun for a player to achieve her goals [10]. This definition also provides a suitable setting for a gaming narrative, since game plots usually revolve around some kind of conflict. Finally, product quality assurance, i.e., testing and adjusting a game title, is a difficult task that demands suitable tools for understanding gameplay such as visualization [14] and even a complex player experience evaluation [15].

B. The Metroidvania Genre

Some classical titles of Metroidvania games are *Castlevania: Symphony of the Night*, *Super Metroid*, and *Chasm or Axiom Verge* [6]. These titles are remarkable water divisors of this genre, so it is important understanding the history behind the birth of Metroidvania genre.

In 1986's Metroid, the first game of the acclaimed franchise, introduced players to the concept of a big world made of bidimensional platforms [11]. In 1994, the Super Metroid refined that gameplay immensely while increased the complexity of their stories. Three years later, 'Symphony of the Night' expanded the same exploration formula in others directions, adding complexities and depth each turn [11].

It is important to note that Metroidvania is strongly linked to new productions, especially in indie games. This is mainly due to the underlying game mechanics constitute a basis for new productions, and titles make use of enthralling mechanics like: exploration, character improvement, platforming, and combat [11]. Koji Igarashi, known as the creator of Castelavania, was surprised by the emergence of the term Metroidvania[4]. He comments that a 2D platformer based on Zelda also refers, in some way, to Metroid, since the intention behind Castlevania was to produce a game that referred to the iconic Zelda.

In a typical Metroidvania, the lack of clear direction in gameplay encourages scenario exploration and at the same time forces the player to revisit the path she has gone through.

[4]https://www.usgamer.net/articles/gdc-2014-why-koji-igarashi-is-grateful-for-the-word-metroidvania

According to Wahlberg [16], this "kickback" offers little variability with differences in some scenario parameters, such as difficulty, types of enemies, and how waves of enemies are fired. Wahlberg developed a study based on the framework of the MDA Framework and its types of fun [17]. He concluded that the concept of recurring scenario exploration, entitled *backtracking level design*, is one of the pillars of the Metroidvania genre.

According to [18], the term Metroidvania was coined by Jeremy Parish and Scott Sharkey [5]. These are the main are main characteristics of this genre[6] [18] [10] [16] [11] [19]: scroll is allowed in both horizontal and vertical axes; players are tasked with finding and pursuing in-game goals; Access to the scenery is limited by inventory and skills, so maps are therefore gradually and recurrently exploited until players find items such as keys, doors, and heights, or acquire abilities such as a high jump, flying, and gliding; the gameplay rewards players for their creative ways of moving on the map, even if they seem to deviate from the overall purpose of the game; the game offers items and abilities as the player advances through the scenarios and story. These power-ups reinforce the sense of progress; as the player progresses in the game in terms of options for exploration, her curiosity is poised to revisit locations in order to try to discover secrets, passages, items, and challenges using her newly discovered ideas; and despite the focus on exploration, this type of game does not limit its ability to tell a story.

C. Requirements of Metroidvania Games

Based on opinions from the critic, we can summarize the main features of a Metroidvania: narrative; extensive and well-designed maps; evolution of the character; and fair balance of difficulty. Analyzing these requirements from a technical perspective, in the process of maintaining narrative immersion with such elements, efforts are needed to optimize key aspects of the game. For example, all of the content in the extended scenario may not fit at once in memory, so the genre demands intelligent management of characters, textures, meshes, and animations. This is much like memory paging in operating systems.

On the other hand, scenario loading should occur in *background* so the impact of this on gameplay does not cause an immersion break that harms user experience. NPCs, in their tur, need general behaviors that work in game scenarios while being able to react to the player's avatar. In addition, AI should be conveniently integrated for allowing the behavior customization [20]. Other elements such as battle system and alternating skills are also important for games of this genre.

D. Existing Solutions for Metroidvanias

We will now discuss the applicable solutions found through bibliographic research. We will then relate how each solution performs according to the previously established requirements.

[5]https://www.wired.com/2007/03/bonus_stage_deb/

[6]http://gaming.wikia.com/wiki/Metroidvania

1) ***Search Methodology:*** We searched for similar works with the objective to delineate a framework for the genre Metroidvania. Only a few scientific articles on the subject were found even by trying several combinations of keywords in many search engines. All attempts included the term "metroidvania".

An important inclusion criterion consists of dealing with game development from a software perspective. So, we we hoped to identify articles addressing technologies, components, and software architectures. Given the small contingent of papers found, the exclusion criteria selected consisted of not dealing with aspects of game programming. Most results referred to *game design*, plus sociocultural and market analyzes.

The best results were obtained with the *Google Scholar* and the combination of the terms "metroidvania" and "toolkit", which resulted in only one article. The combination of "metroidvania " and "game engine" produced 48 articles, most of these are related to Procedural Content Generation (PCG) such as ANGELINA [18]. Similarly, Kärkkäinen [21] addresses the use of procedural generation of scenarios for Metroidvanias. This dominance of the PCG theme in academia is reasonable. given the dimensions of maps, missions, and complexity of the genre, it is interesting that developers rely on "automatic suggestions" when creating content [2].

2) ***Solutions Found:*** We found works that approached the genre Metroidvania, but applied to other problems. Due to space limitations, we will limit ourselves to discussing only the works that most closely match the scope of our research. Rodríguez [22] developed a multiplayer Metroidvania for the Web using JavaScript on top of *Clockwork.js*[7] engine. In addition to the challenges faced in the genre, the game features two characters with different control modes. She states that the game lacked careful planning to be optimized for a better online experience. However, this work contributes with an elucidation of the rationale underlying the genre.

Kay and Powley [23] studied the influence of pathfinding methods and their respective visualization on the exploratory behavior of players. They concluded that the aspect of NPC navigation, implicit in the implementation of a 3D Metroidvania, is capable of impacting the game's visual. Therefore, navigation can provide useful insights for game designers. Daly [24] reports the development of a Metroidvania using the open source engine and discusses the following game systems: movement, slopes, animation, combat, power-ups, a custom "sanity" system, and visual effects. Daly adopts finite state machines for a simplistic AI, but also discusses refactoring: he felt sabotaged by himself during implementation. This demonstrates the importance of adopting a framework.

Büsser [25] addresses the challenge and implementation details of a Metroidvania. Comparing the present proposal with what this author has developed, we conclude that our algorithm of pathfinding provides a more complex behavior than the simple patrolling provided by Büsser's implementation. For

[7]http://clockwork.js.org/

example, our framework supports flying NPCs. Minkkinen [26] talks about the lessons learned from the development of platformers. The author considers his final result as a very "raw" product and it is essential to keep the project simple, so the author has dedicated himself to improving the central part of the project more and more. In effect, the fluidity of the movement of the character and the camera were prioritized instead of adding new functionalities and mechanics to the game.

Nakamura and Câmara [6] proposed guidelines for evaluating the "fun" of exploration games based on the game design definitions of Jesse Schell [10]. These authors analyze observation experiments and analysis reports from the player's experience to present suggestions on the requirements, and what game design practices may better suit this type of game. In her research project, Maciejewski [27] describes the experience of creating a game of the genre Metroidvania using Python and Pygames.

Our investigation on the existence of frameworks especially designed for the development of Metroidvanias suggests a lack or deficit of existing solutions. This finding corroborates the motivation of the present study to propose an architecture and its scientific contributions.

III. Proposed Framework

We start this section with an overview of our framework architecture. For a better understanding, this section was subdivided into the main problems and the respective solutions provided by our framework.

A. *Overview*

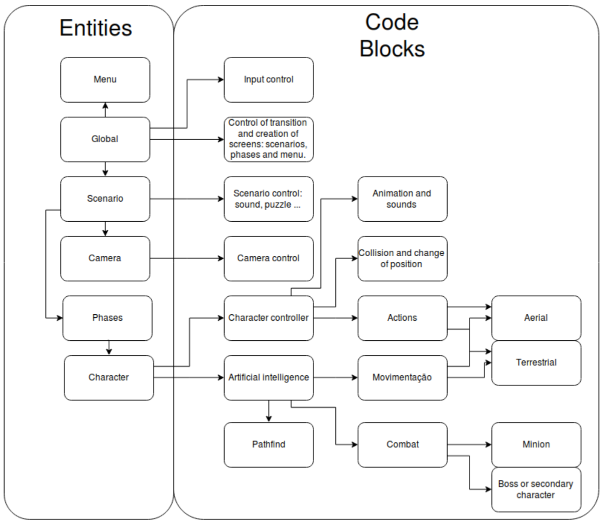

Fig. 1. Overview of the proposed framework.

Our architecture is composed by two main components: entities and code blocks (see Fig. 1). Entities have both attributes and codes. Code blocks, in their turn, encapsulate the processing logic and can be subdivided into smaller blocks responsible for specific functions. Code blocks can be adaptable to entity attributes, so depending on the attributes, a different processing may be triggered. Block verification is performed in creation time. Moreover, this component can be assigned dynamically, i.e., instances can be re-assigned to entities at little to no cost. An optional verification may when all blocks are loaded from a context, which gives space to more sophisticated code blocks and entity allocation algorithms.

We also define a Global entity that has control over what will be displayed on the screen. This entity also stores all persistent game state variables such as life, location, and so on. The Scenario entity is composed by Phases, so performance improvements can be obtained by subdivision [20]. Moreover, this approach also prevents this huge object from being totally loaded.

Phases contain Characters, which are complex entities assigned to two code blocks, one for control and the other for AI. The Character Controller is responsible for interaction of character which is called by AI or by user inputs. Character Controllers are instantiated according to the demand: for example, aerial or grounded characters defines what block will be assigned to that entity. It is worthy to observe that this component usually serves NPCs in the game logic layer. Moreover, since our architecture decouples entities from code blocks, a flying character may change its type during gameplay by assuming an alternative grounded form, for example.

The AI code block divided into Pathfinding, Combat, and Steering blocks. The Character Controller is invoked every time the system needs to change character behavior. The Combat block, in its turn, is subdivided into minion and "boss" code blocks. We adopted this approach because bosses and secondary characters usually consume more resources since these NPCs display a broader variety of attacks and resort to more sophisticated AI methods.

B. *Game Map Management (GMM)*

We developed algorithms to avoid overloads by dividing the map into Scenes. This process is applied recursively, thus subdividing Scenes into Phases. Phases are loaded according to a position of the player in the map. Moreover, we also adjust the player position relative to the current level in order to avoid floating point issues.

At first, we limit the loading to the characters, code blocks, and other information that will be persistent until the player "touches" the border of another scene. When this happens, the framework loads metadata, characters, and code blocks of the potentially new current scene. Phases will also be managed, since their code blocks are responsible of puzzle management, playing ambient sounds, triggering enemy wares, and other specific gameplay behavior. Similar to Scenes, Phases are loaded according to the distance traveled by the player's avatar. Since our focus is on platformers, we resort to an efficient neighborhood lookup over a hierarchical grid for determining which Scenes and Phases are in the vicinity of the current view. Finally, Phases and Scenarios are kept in an active list, so these unloaded when they are not reported by the hierarchical grid search.

C. Interaction Between Characters and Environment

Interactions are divided into movement, i.e. steering, and interactions between characters. Regarding movement, we have the situations of collision between a character and other entities. These can be another character or even map elements. Detecting these combinations in terms of entity attributes allows us to handle each type of collision properly.

For example, we first chose that protagonists would not collide with each other in our prototype game. This behavior was also applied to NPCs later, since users found this useful to avoid excessive stress when controlling characters and to make the game more fluid, despite such choice would have an impact on the immersion. However, such loss would not be noticeable by most users in our focus group. Moreover, we decided to delegate the movement's handling to specialized entities, such as fixed and moving platforms, since this would facilitate future maintenance and addition of new steering behaviors.

Entity interaction, in its turn, allows both entities to handle interaction in the same event. For example, during a combat, one character takes damage and the another increases its sum of hits struck. It is worthy to mention that this situation occurs as a consequence of reported collisions between pairs of entities, therefore both will receives the contact information and trigger their corresponding code blocks. This is essential for effective AI programming, since such controller needs feedback information about any decisions taken. Characters in general also need such information to know what actions to trigger, what animation to perform, etc.

It is important to highlight that our implementation sorts collision handling code blocks before their execution based on entity type and identifier. This is an essential step for obtaining stable code execution to avoid intricate bugs, otherwise unpredictable behavior will be displayed during gameplay.

Finally, another important scenario regards player inputs, for example, to push or pull a lever. The lever is an interactive entity that does not move and then executes a specialized code block for supposedly receiving damage: this block will actually, trigger events and scripts responsible for managing puzzles, quests, etc.

D. Character Recycling

To improve performance, characters keep their global position even in loading time. This simple information is sufficient for our recycling algorithm. Characters discarded due to death or even leaving the borders of the screen view area are not unloaded immediately, but left disabled and collected into a list. So, these resources do not demand a new verification step and can be reused in the future. It is worthy to mention that any persistent information about these characters are saved to Scenes, Phases, and Global scope before actual recycling. This happens because such entities' attributes must be reset to their default values. In the case of restarting a map region, Scenes and Phases are responsible for return any "living" characters to their standard position and restore their initial state attributes, when necessary.

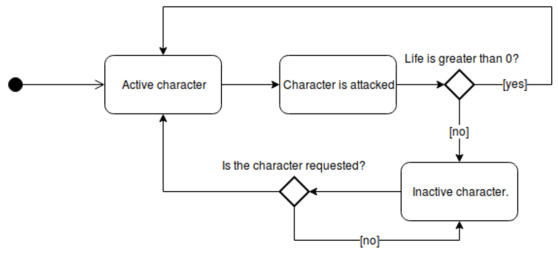

Fig. 2. Character recycling diagram.

E. Pathfinding

When faced with the problem of moving NPCs in a scenario, oneself should think of trying to produce a balance between the best and faster paths [12]. This is because, unfortunately, this problem has a tendency to arrive at unacceptable response time scales for a game, especially when a large number of possible routes are found [28].

It is difficult to devise an algorithm that can fully understand the trade-off between these two goals, so the most reasonable option is to define which aspects are the most important at the context. Starting from this assumption, we must define how to represent the environment to be used in the pathfinding algorithm. A full grid of NPC routes can be prepared beforehand [12] [28]. However, this means additional work for level designers. PCG methods may also present problems and limitations in this regard [19].

According to Michael Cook [19] PCG for video games is a crucial area of research and development in the industry. Assorted, high-quality materials are required in large quantities, which continues to pressure content developers. At the same time, PCG is gaining important influence on the culture of video games. Games that explore such ideas and adopt some kind of PCG are usually well-received by critics [19]. Thus, the use of pathfinding that does not use a preproduced grid goes beyond diminishing the work of the level designer, and can even adapt to procedural systems and influence the satisfaction of the final user.

Adopting collision sensors is another, feasible option available and related to level design. According to [29], Point Content sensors can mark points in the environment and verify the collision. From that, it is possible to identify objects in the scenario near the area around the character. The Line Trace sensor, in its turn, the NPC produces lines to determine its position from collisions with other the objects. This technique ends up creating an arc, i.e., it is tries to move around obstacles. This sensor can be used to identify collision only on objects in the character's field of view.

Finally, Collision Detection sensors [29] identifies contacts in the character's body, so that the character position is only identified when there is a collision. Another possibility, which is not being mentioned Champandard's book [29], resorts to area collision: all the elements colliding with a rectangle comprising the camera are reported.

Starting from these initial options, those that best fit the

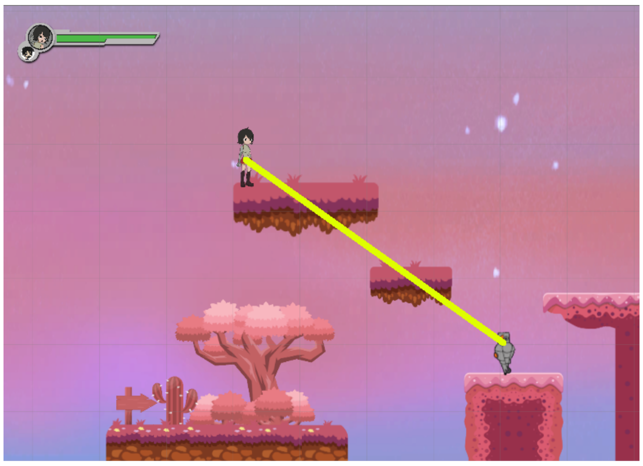

Fig. 3. Line traced between a character and his enemy (target). The algorithm considers the existence of platforms for navigation.

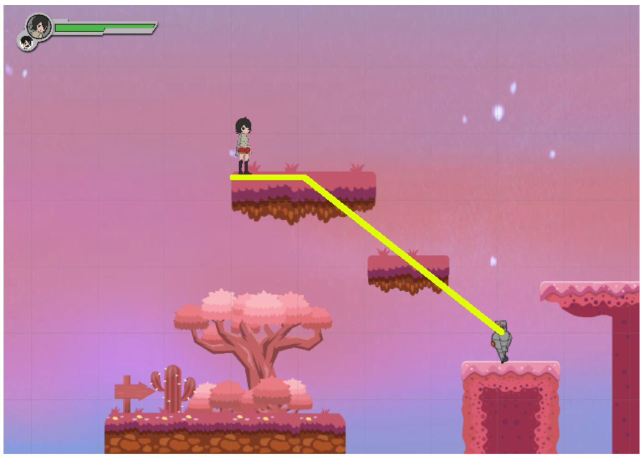

Fig. 4. Line traced based on points returned by pathfinding.

aspects of the Metroidvania genre were selected. Point Content is not considered suitable. First, there is the possibility of not finding some platforms, which can cause unexpected behavior of the NPC. Second, it may incur an unnecessary overload because many scene objects that would not be used for the pathfinding algorithm are returned. Collision Detection was also discarded since the sensor area only considers the characters' bodies, which ends up returning useless objects. So, we found Line Trace is the most appropriate sensor because it contains only the objects that are between the character and its target, thus avoiding both processing and memory overheads.

After the environment mapping is performed, we should use an algorithm that uses these objects not as obstacles, but as a support that helps to bring the NPC closer to its goal. Therefore, this object must not be avoided. Another important aspect is that not all supporting objects need to be used: there would be extra of effort for a simple jump, for example.

Our pathfinding algorithm receives as parameters the physical limits applicable to the character to seek. Whenever possible, the platforms closest to its target are found. In this way, it is possible to make the most of the energy at each jump. This, also makes it easier for the character to reach his destination in less time. This approach has technical advantage and also extrapolates the best use of the resources: the visual result is more pleasant to players; there is a greater "sense of intelligence' by avoiding very segmented routes; and it abstracts grids of colliders loaded dynamically.

F. Combat System

The combat system is responsible for managing the battle actions of players and NPCs. Each attack, defense, and reaction is implemented at the *frame* level, much like traditional fighting games. This decision was made to allow a fine-grained control of the behavior of the characters while the battle unfolds. Such granularity is important for applying any balancing fixes or even fixing *bugs*. In addition, this system allows to form produce *combos*, i.e., assembling combinations of attacks in the same sequence.

Each character has a list of possible actions, chosen according to the player's controls and current ability tree. The damage is computed on a per-frame basis to prevent loops, which may lead to infinite combos, and to allow damage scaling as the combination progresses. This opens up possibilities for more elaborate combat strategies, such as applying *counters*, *resets*, and *setups*. We adopt combos trees for the sake of diversity: the combat system should looks for valid sequences instead of repetitive and boring fixed sequences.

G. Character Tagging

Tagging mechanics allows character control to act on an auxiliary character and creating strategies for differentiated game experiences. This mechanic requires an AI capable of possessing 2 entity groups: allies and enemies. When character meets one of the groups and attack the other. Given our code blocks approach, a single AI instance can deal with both groups. In addition, this mechanism is also aware of the options of the opposing group.

For the exchange to be activated, the character A that is being controlled by the player will have its AI turned on, while its companion character B will have its AI enabled. The player activates the switch by commanding B and deactivating its AI as the opposite is done for character A. Another way of doing the exchange consists of, starting from the previous situation where A has AI activated and when called through a global command, B keeps its AI disabled and A, which previously had AI activated, changes to off and then starts receiving commands. In this way, character B stands still.

Tagging characters increases the chances of playing in different situations. Each character may have unique abilities to be explored both in combat and in puzzles. For example, a puzzle may require to keep a character stopped or activating a given ability, which in turn increases the depth of gameplay.

H. NPC Behavior System

Tagging characters demands AI-controlled behaviors, which must be created in order not to interact directly with the character, but to constitute an outer layer that calls the interaction methods found in code blocks assigned to that character. This process of abstraction depends on action ids, so normal player commands are also given to characters in this same way. We decided to split AI into combat and movement aspects to simplify the creation of behavior for NPCs.

1) Movement Control: To better understand movement, it is necessary to plan how the environment is populated. As the interaction with the scenario will also be provided as a feature, this requires the creation of platforms. In this genre, it is convenient to have two types: the ground type and the aerial type.

Platforms can be accessed by jumping over them, because their physics do not allow the character to cross and fall. However, there are differences between aerial and grounded platforms. Air platforms allow characters to trigger defensive and jump actions simultaneously when the character leaves that platform. This is not allowed for grounded platforms. A similar difference occurs for flying characters, since no interaction is required with aerial platforms in this case.

Therefore, the steering algorithm presents a division between terrestrial and aerial characters. This is necessary since there are different interpretations of obstacles and these are used. Aerial steering interprets ground platforms as obstacles that must be diverted or circumvented. This happens because grounded platforms do not allow flying characters to move freely within it. Conversely, aerial steering do not interpreted aerial platforms as obstacles. In grounded steering, in its turn, aerial platforms are interpreted as obstacles that are possible part of a route. This allows movement in all directions. Regarding the ground type platforms, it is not possible to move down across.

The steering behavior is computed based on the points returned by the pathfinding algorithm. So, the resulting behavior contains target navigation points that must be traced, besides the set of instructions indicating which commands the character must receive so that, from the environment in which it is placed, it is able to reach the goal by restricting its movement towards the target. Once the first target point is reached, the global coordinates for the next navigation point should be removed from the list. This process is repeated until the list is empty or the global target is reached. This last condition is determined by a minimum distance, which will be similar to the distance required for the NPC to attack.

Moreover, the coordinates must also be translated into directions for the sake of animation. If the target is to the left or right of the character, it must change, pass as parameter the changed horizontal directional to the move function. For the vertical direction, if the target is positioned above the character he should jump and if he is below should put in a position of defense and jump thus making the character descend from the platform.

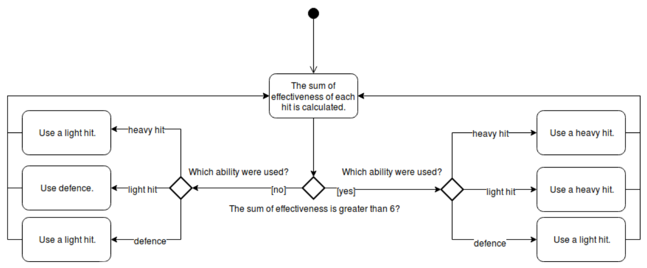

Fig. 5. State machine describing, at high level, how the combat system works.

2) Combat Control: The combat system provides two types of AI for battles, one being simpler and the other more advanced, so these mechanisms can be used in different NPCs. The simple AI is used for ordinary minions and the more advanced is better suited for bosses and players' companion NPCs. Bosses and special NCPs display a greater variety of combos and attacks, in addition to having a more elaborate combat strategy.

For the intelligence of the minion, the combat AI behavior comes down to sending the information to the code block responsible for the interaction of the character. An attack will be thrown whenever a target is within range. The actual attack and the corresponding animation depends on the character being controlled, plus its ability tree. The boss AI is far more complex. First of all, this happens because two different types of attacks must be managed during combat, i.e., light and heavy blows. In addition, it must properly handle animation when receiving damaging attacks. Moreover, bosses can also resort to defense.

In both AIs, the overall combat algorithm should cause NPCs to take actions according to the effectiveness of previous actions. That is, it should obtain a good combination of moves so that it has a greater effectiveness in its attacks.

Such verification is obtained by a sum representing a score of 0 to 7 of the effectiveness of the action. Each factor in this sum is not cumulative. The damage is counted as follows. When the character receives no damage, the sum will be incremented by 5. Another increment will be given to hit an enemy. Two points can be obtained by hitting the target with the open defense or hitting it in defense with a heavy blow. This adds pressure to the attacked character. One point may be obtained by hitting an enemy who is not the target with the open defense or hitting him in defense with a heavy blow. If the hit does not hit or is reflected, no point will be received.

From the result of this sum we can get a suitable reaction. If the sum is 6 or 7, the stroke will be maintained until the 3rd round of the same stroke, in which case the action will be modified if the action in question is a light blow. In that case it will be a heavy blow, but if it is the defense action then it will be the light blow. If the sum is less than 6 the action will be modified if it is the heavy blow it will be the light blow, if it is the light blow then it will be the defense, and if it is the defense then it will be the slight blow.

IV. Implementation Aspects

The Unity 3D game engine and the C# language were chosen for the implementation of the framework. C# is a powerful, expressive language which supports reflection and other sophisticated features of object-oriented programming. Unity 3D was chosen due to its learning curve, documentation, community support, and free license. Development took 2 months. We resorted to Unity's debugging console, visual editor, and game preview mode during development.

Some serious limitations were found and resolved in the early stages implementation. For example, jumping was handling collisions in the character entity. We limited user tests before presentation to the focus groups. For example, our estimate of the time required for solving the puzzle was based on our own knowledge of this mechanic.

It is necessary to emphasize that the parts that consumed the most time of implementation were *pathfinding* and artificial intelligence, plus debugging. Both aspects have to solve complex problems with generic algorithms for the sake of reuse. We found it difficult to analyze results we obtained. For example, after some tests the tester IA starts to get used of the strategies displayed by NPCs. Consequently, the game seems to be simpler and easier than it actually is for the general audience. In the case of pathfinding, we resorted to visualization in order to understand each pathological case of our algorithms. We decided to endow entities with visual elements, i.e., gizmos in Unity 3D, by including their steering path and then tracing lines with the route points that were obtained with the algorithm. This information was gathered across frames, so we could effectively analyze it.

Regarding AI, we had to keep a lot of visual elements available in the screen besides the colliders. We also resort to video recordings of the interaction and frame-to-frame playback for more effective debugging. Certain situations demanded magnification of specific object and gizmo parts in the recorded videos. These were played in slow motion, in addition to put a key to adjust the speed between slow and normal. Moreover, testing time also demanded visualization of the variables and their behavior over time.

V. Evaluation

The prototype was evaluated by resorting to both experimental performance evaluation and a focus group.

A. User Evaluation

Participants introduced to general information about the prototype and played a test map. Some participants were developers which were surprised by the time spent in both prototype and framework development. When asked about the performance and the overall score of the game, the participants assigned a maximum score on a scale ranging from 0 to 10. We also collected improvement suggestions.

Players were invited to reassess the game once these changes were incorporated. This procedure helped us to spot and fix bugs in character movement: avatars could cross walls during some jump situations. This particular bugfix provided us with new ideas to increase the game logic layer. Initially, we calculated that players would take around 10 minutes completing the map. However, all players ended up spending more time because of the challenge found in the puzzle. According to players' comments, the worst part was the unbalanced, hard to fight boss AI and the time to master the character tagging mechanic. We studied the hypothesis of a controller based on the boss's life. This the combat AI block can be configured or swapped to make more "smart" decisions, thus enabling programmers to change the pace of the battle. We advocate this sophisticated behavior helps to reinforce the emotions transmitted by game lore on the boss.

B. Performance Evaluation

Tests were carried out in a computer with i5 3330, 6GB RAM, and XFX AMD Radeon R7750 1 GB DDR5 GPU running Ubuntu 17.04. First we tested the game using all possible resolutions. The prototype game kept running at 60fps.

We performed stress tests on the average FPS versus the number of active NPCs and interacting constantly (see Fig. 6). Up to 50 NPCs do not cause any overhead in our prototype. Then, the FPS drop rate remains constant up to a total of 125 NPCs, where we have approximately 20fps. Similar behavior is obtained from 175 to 200 NPCs when average FPS stabilizes around 10. Metroidvania titles rarely display more than 30 active enemies, thus our prototype game achieved very satisfactory performance.

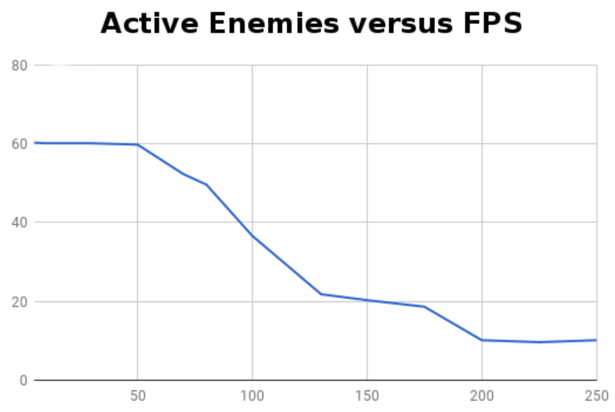

Fig. 6. Enemies versus frames per second.

We tested the GMM by traversing the 4 Phases of a Scene. We collected the RAM used by the game with GMM turned on and off (see Fig. 7). GMM displays lower initial RAM usage, which is even lower in the second phase, but increases decreases in the third phase and is lowered again as we move to the fourth phase. This decrease makes sense since there are three parts loaded at that point. Consequently, our framework can fulfill the purpose of reducing memory consumption.

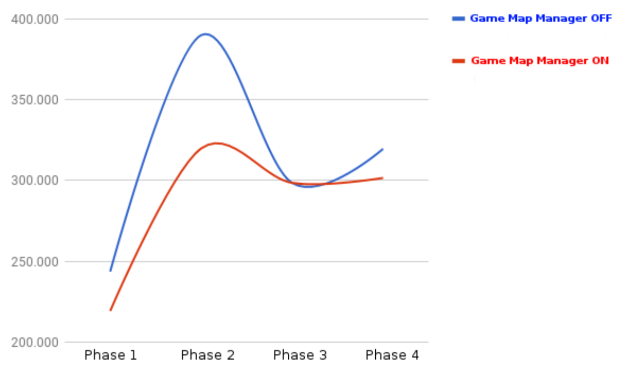

Fig. 7. RAM memory, in MB, used during gameplay over 4 Phases. Better viewed in color.

VI. Conclusion

In this work, we proposed programming framework for supporting the development Metroidvanias. We implemented a framework prototype that abstracts many complexities found in real world projects by proposing solutions to common problems found in this genre. We developed a game prototype with the purpose of verifying our framework prototype. Experimental evaluation demonstrate that our framework is both feasible, useful, and efficient.

Future research opportunities starting from this work includes: adding support for item dropping and pickup; incorporate procedural content generation techniques; adopting AI with a predefined sequence of actions to increase the range of implementation options; and improving AI for finer control over bosses, for example, reinforcing their emotions and roles in the narrative.

References

[1] J. P. d. P. R. d. Silva, J. Veiga, and C. V. d. A. Carvalho, "Desenvolvimento de jogos utilizando xna: um exemplo com o jogo spacex," *Revista Eletrônica TECCEN*, vol. 5, pp. 59–70, 2012.

[2] A. d. O. d. R. Franco, J. A. de Mesquita Neto, J. G. R. Maia, and F. A. d. C. Gomes, "An interactive storytelling model for non-player characters on electronic rpgs," in *Proceedings of SBGames 2015*, 2015, pp. 33–41.

[3] J. Gregory, *Game engine architecture*. AK Peters/CRC Press, 2017.

[4] P. Schuytema, *Design de games: uma abordagem prática*. Cengage Learning, 2008.

[5] F. Boaventura and V. T. Sarinho, "Mendiga: A minimal engine for digital games," *International Journal of Computer Games Technology*, vol. 2017, 2017.

[6] R. Nakamura and P. G. Câmara, "Design de jogos ea experiência de exploraçao de espaços," *Obra digital: revista de comunicación*, no. 5, pp. 20–35, 2013.

[7] T. W. Stalnaker, "Procedural generation of metroidvania style levels (thesis)," 2020.

[8] A. G. Rodríguez, C. Cotta, and A. J. F. Leiva, "An evolutionary approach to metroidvania videogame design," in *XVIII Conferencia de la Asociacion Española para la Inteligencia Artificial*, 2018, pp. 518–523.

[9] Z. A. Algfoor, M. S. Sunar, and H. Kolivand, "A comprehensive study on pathfinding techniques for robotics and video games," *International Journal of Computer Games Technology*, vol. 2015, p. 7, 2015.

[10] J. Schell, *The Art of Game Design: A book of lenses*. CRC Press, 2014.

[11] C. Nutt. (2015) The undying allure of the metroidvania. [Online]. Available: http://www.gamasutra.com/view/news/236410/The_undying_allure_of_the_Metroidvania.php

[12] G. S. David M. Bourg, *AI for Game Developers*. "O'Reilly Media, Inc.", 2004.

[13] G. L. Zuin, Y. Macedo, L. Chaimowicz, and G. L. Pappa, "Discovering combos in fighting games with evolutionary algorithms," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 2016, pp. 277–284.

[14] V. R. Feitosa, J. G. Maia, L. O. Moreira, and G. A. Gomes, "Gamevis: Game data visualization for the web," in *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, SBC. IEEE, 2015, pp. 70–79.

[15] A. S. Bastos, R. F. Gomes, C. C. Dos Santos, and J. G. R. Maia, "Assessing the experience of immersion in electronic games," in *19th Symposium on Virtual and Augmented Reality (SVR)*, SBC. IEEE, 2017, pp. 146–154.

[16] T. Wahlberg, "Blockades in the metroidvania genre of games: A examination of backtracking," 2015.

[17] R. Hunicke, M. LeBlanc, and R. Zubek, "Mda: A formal approach to game design and game research," in *Proceedings of the AAAI Workshop on Challenges in Game AI*, vol. 4, no. 1, 2004, p. 1722.

[18] M. Cook, S. Colton, and J. Gow, "Initial results from co-operative coevolution for automated platformer design," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2012, pp. 194–203.

[19] ——, "The angelina videogame design system—part i," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 2, pp. 192–203, 2016.

[20] R. Nystrom, *Game programming patterns*. Genever Benning, 2014.

[21] J. Kärkkäinen, "Proseduraalisesti generoidut metroidvania-kentÄt," Master's thesis, Tietojenkäsittelyn koulutusohjelma, 2016.

[22] S. Barbero Rodríguez *et al.*, "Desarrollo de un videojuego sobre un motor javascript," B.S. thesis, Escuela Politécnica Superior, Mayo 2017.

[23] M. Kay and E. J. Powley, "The effect of visualising npc pathfinding on player exploration," in *Proceedings of the 13th International Conference on the Foundations of Digital Games*. ACM, 2018, p. 60.

[24] C. DALY, "Utilizing modern game design practices in a solo-dev environment," Ph.D. dissertation, Stetson University, 2016.

[25] T. Büsser, "Creation of a video game the intricacies of game development," Master's thesis, Kantonsschule Sargans, 2018.

[26] T. Minkkinen *et al.*, *Basics of Platform Games*. Kajaanin ammattikorkeakoulu, 2016.

[27] S. D. Maciejewski, "Shapeshifter of py," in *SUNY Undergraduate Research Conference*, 2015.

[28] A. U. ARAMINI, "An ai assisted framework for the design of 2d platformers," Master's thesis, Politecnico di Milano, 2017.

[29] A. J. Champandard, *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. New Riders, 2003.