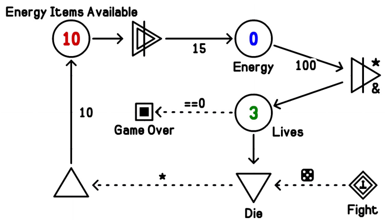
Game Mechanics Design: Applying Machinations to Eliosi's Hunt

Tiago Zaidan[1]* Daniel Zaidan[2] Luís Fabrício W. Góes[3]

TDZ Games[1]
PUC Minas, Computer Science Department, Brazil[2]



Abstract

The game mechanics design process suffers from the lack of widespread formal methods. Game designers tend to build prototypes to test their mechanics before implementing them in a game. The current methods of prototyping mechanics can be time-consuming and often do not provide a model that has a satisfactory trade-off between the level of abstraction and accuracy. The Machinations Framework was created to address these issues as a method to formalize, model, prototype and test game mechanics. The framework makes use of a designer-friendly node-based model of a flow of resources and an online tool to simulate the diagram, making it easier for designers to evaluate their mechanics. Machinations is most used to model a game's internal economy, which is a type of mechanic, along with physics, progression mechanisms, tactical maneuvering and social interaction. The tool's iterative nature fits perfectly a game designer's need to constantly change and improve one's designs. This paper presents the use of the Machinations Framework to design the internal economy of the game Eliosi's Hunt currently being developed by TDZ Games. The use of the framework allowed us to visualize the game's internal economy's structure and flow of resources. It also showed the economy's feedback loops and emergent behavior, possible deadlocks, the impact of random values and its balance in order for the designer to improve it through fast iterative cycles.

Keywords: Game Design, Game Mechanics, Machinations Framework, Game Mechanics Design, Eliosi's Hunt, Feedback Loop, Internal Economy, Game Systems.

1 Introduction

A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome [10]. When it comes to digital games, however, the term game mechanics appears more often than rules [3]. While rules are the explicit and unambiguous instructions shared by all players [10], game mechanics are more detailed and often hidden from a player. According to [3], the rules of a game consist of the information players need to play a game, for example, in a platform game "press Space to jump" is a rule. The outcomes of the rule, such as jumping on enemies or falling and dying, are mechanics. Therefore, mechanics include everything that affects the operation of a game and what is needed to program a game [3].

In [7], it was presented a formal approach to understand games and improve the design process of games. The MDA model divides games in three levels: Mechanics, Dynamics and Aesthetics. A designer and a player approach a game from different perspectives. A player experiences a game through its aesthetic, which represents the pleasures games can provide, such as sensation and discovery [7]. Although the aesthetics are also the designer's main goal, he can only directly design the mechanics, from which dynamics will emerge. In turn, it is the dynamics (the run-time behavior or game systems) that will evoke the emotional responses of the aesthetics. The MDA framework describes the relations of the three levels but lacks a tool to build each of them.

The Machinations framework was created as a formal method to view, understand and design game mechanics as a flow of tangible and abstract resources [4]. Previously, there were three main methods of testing and prototyping game mechanics [3]. Software prototyping represents faithfully the mechanics but are usually time-consuming to build and require code skills. Paper prototyping, a non-computerized tabletop game created to test aspects of the mechanics, are faster to build but it fails to represent mechanics which are dependent of heavy calculations, such as precise timing of physics. The last technique is physical prototyping, which requires the designer to draft rules and play a game in real life. While it is even faster and more adaptable than paper prototyping, it is a hard task to accurately adapt the mechanics to be tested in this scenario. The Machinations framework allows designers to represent and test the mechanics in a way that is designer-accessible and yet maintains the structural features and dynamic behavior of a game. These features, such as feedback loops, are presented in a way which is clearly visible in order for designers to better communicate, iterate and construct emergence on a game mechanics' design [4].

According to [2] there are five types of game mechanics: physics, internal economy, progression mechanisms, tactical maneuvering and social interaction. Machinations focuses mainly in a game's internal economy, that is, the transaction of game elements which are collected, consumed, and traded [3]. The resources include anything from money, energy and ammunition to enemies. There are also intangible resources (that does not exist in a game world), such as health points; and abstract resources (that does not exist in a game as a whole but only in its current state). For example, in a game of chess, a player might sacrifice a piece to gain

*e-mail: tiagozaidan@gmail.com

strategic advantage. In this case, strategic advantage is an abstract resource [3].

In order to improve the game mechanics's design process, we used the Machinations framework in the development of Eliosi's Hunt. Currently being developed by TDZ Games [1], an independent video games studio based in Belo Horizonte, Brazil, Eliosi's Hunt is a top-down 3D action game, combining the shooter and platformer genres. It is a challenging skill-based single-player game in which the main character, Eliosi, is a young alien obsessed with the idea of becoming a bounty hunter but completing his first contract will be more difficult than it seems. The game depends heavily in physics mechanics such as jumping, shooting and dashing. Another very present game mechanic type is the internal economy.

Eliosi's Hunt internal economy deals with a number of different resources such as ammunition, life points, speed boosts, enemies and most importantly, the energy system. Throughout the game, Eliosi will find several items that give his companion drone energy when destroyed (each level has a unique item associated with it). He can also gain it from killing enemies and destroying checkpoints. The energy goes through a formula which randomizes its amount and takes into account the number of player deaths: the more a player dies, the more energy he will get from the items, acting like a negative constructive feedback loop.

The main goal of this paper is to model and test the Eliosi's Hunt internal economy during the design process through the use of Machinations Framework. By using the Machinations Framework, it was possible to easily identify deadlocks and iterate to create the best solution. It also made the feedback loops of the game system very clear. By observing the action and interaction of feedback loops and its characteristics, it was possible to easily understand and create complex systems as well as balance them. The main contribution of this paper is to show all the process of designing a real game's mechanics through the modeling of its internal economy using the Machinations Diagram, as well as the aspects a game designer should observe and iterate on when creating game systems.

This paper is organized as follows. Section 2 presents the background, elucidating the main concepts and related work. Section 3 and 4 presents the game Eliosi's Hunt and Machinations Framework, respectively. Section 5 presents the application of Machinations to Eliosi's Hunt. Finally, Section 6 concludes and discusses future work.

2 Background

2.1 Game Mechanics

Game mechanics are the core of a game, they are the interactions and relationships that remain when all of the other game elements are removed from a game [9]. The mechanics describe, at the level of data representation and algorithms, particular components of a game. They are the various actions, behaviors and control mechanisms afforded to a player within a game context [7].

As many other process in game design, it does not exist a universal taxonomy of game mechanics [9]. In [3], it is described five different types of mechanics. The physics mechanics define the science of motion and force in a game world. They can be found in games when characters move from place to place, jump up and down or drive vehicles. Physics plays a large role in many modern games from ultra realistic first-person shooters to physics puzzle-games [3].

The second type of game mechanics is the internal economy, that is constituted by mechanics of transactions involving game elements that are collected, consumed and traded. The internal economy of a game typically encompasses items easily identified as resources, such as money, energy, ammunition, enemies and so on. A game's economy is not limited to concrete tangible items, it can also include abstractions such as health, popularity and magical power. The third type of mechanic is the progression mechanisms, which controls a player's progress though a game's level. It is represented by mechanisms that block or unlock a player's access to certain areas, such as levers, switchers and magical swords that allow a player to destroy certain doors [3].

Games have mechanics that deal with the placement of game units on a map for offensive or defensive advantages. This is the fourth type of mechanics, the tactical maneuvering, and it is critical in most strategy games. The mechanics that govern tactical maneuvering typically specify what strategic advantages each type of unit may gain from being in each possible location. The last type of mechanics is the social interaction. Many online games include mechanics that reward giving gifts, inviting new friends to join, and participating in other social interactions [3].

2.2 Internal Economy

An economy is a system in which resources are produced, consumed and exchanged in quantifiable amounts [2]. Games also have their economy, and it consists of the resources that a game manipulates and the rules that govern how they are produced and consumed. A game's internal economy is different from a real life economy, it can include resources such as health, experience, skill, time and units that are not part of a real-life economy, but games can also have money, goods, and services in their economy. To understand a game's gameplay, it is essential to understand its economy, no matter how big or small it is, and creating it is an important design task [3].

A game's internal economy revolves around the flow of resources, and it refers to any concept that can be measured numerically [3]. Anything that a player can produce, gather, collect or destroy is probably a resource. Not all resources are under a player's control, time for example, is a resource that normally disappears by itself and a player cannot change that. Platforms, walls and any other type of inactive or fixed-level features are not resources [3].

Resources can be tangible or intangible [2]. The tangible resources represent physical objects that exist in a game world, they can be stored or transported for example. The intangible resources have no physical properties in a game world, they do not occupy space or exist in a particular location [2]. Medical kits are tangible and health points are intangible resources in shooter games for example [3].

Resources can also be divided as abstract or concrete [3]. The abstract resources are intangibles but are computed from the current state of a game. For example, in chess a player might sacrifice a piece to gain strategic advantage over his opponent and this strategic advantage can be considered an abstract resource. A game usually does not tell players about abstract resources, they are used only for internal computation. It is important to note that not all intangible resources are abstract, experiences points are concrete resources that must be earned and sometimes can be spent like money. Happiness and reputation are two other intangible resources that are concrete parts of a game [3].

The internal economy of a game can also be used to influence a player's progression through the levels [3]. For example, power-ups and unique weapons can be used in an action game's economy. They can be used to gain access to new locations. A double-jump ability in a platform game can allow a player to reach a higher platform that was initially unreachable. In terms of economy, these abilities can be considered as new resources to produce the abstract resource accesses. These accesses can be used to gain more rewards or can be required to progress through the level. A designer should be aware of a deadlock situation and needs to provide means to break deadlocks if one occurs. For example, a game might have a special enemy that can be killed only with an energy gun guard-

[1]available at www.tdzgames.com

ing the exit of a level. This energy gun in located somewhere in the same level and can be used throughout the level. When a player finds the energy gun, it is loaded with bullets and there is no more until the next level, but a player does not know this in the first time playing. If a player spends all his bullets before facing the special enemy, the designer has created a deadlock situation. Thus a player needs access to the next level to gain bullets but needs bullets to gain access [3].

2.3 Feedback Loops

According to [7], a game designer creates the mechanics and they, in turn, give rise to the dynamic system behavior which is the run-time behavior of the mechanics acting on a player's input. The dynamic system leads to a particular aesthetic experience. Interestingly, a player perceives a game through the aesthetics, which is born out in the observable dynamics and eventually, operable mechanics [7]. This means that game design is a second-order design problem [10], because it can only indirectly design the experience by directly designing the mechanics.

As mechanics lead to dynamics (or systems) when a player interacts with a game, it is important to have a great understanding of systems and how they work. According to [10], a system is a group of interacting, interrelated or independent elements that forms a more complex whole. The field of cybernetics was first introduced by Norbert Weiner in his book Cybernetics or Control and Communication of the Animal and the Machine [11]. According to [10], cybernetics is the study of the regulation and control of systems. There are three elements of a cybernetic system: a sensor, a comparator and an activator. The interaction between these elements form a feedback loop [10], in which the information about the result of a transformation or an action is sent back to the input data.

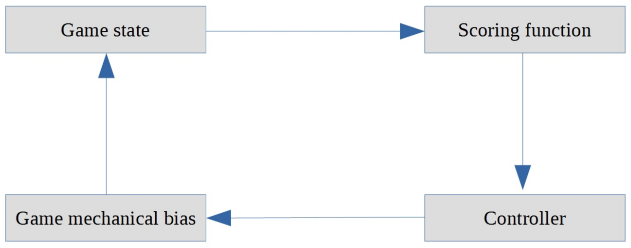


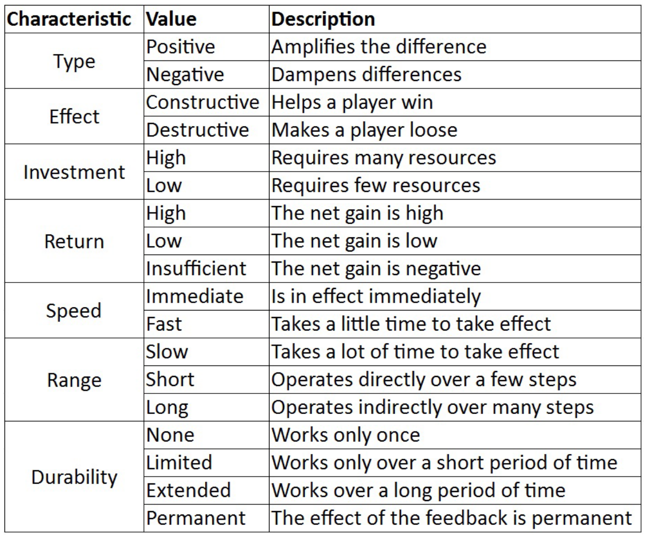
Figure 1: The structure of feedback loops in games [8].

In his 1999 lecture at GDC: "Feedback Systems and the Dramatic Structure of Competition" [8], Marc LeBlanc presented a model to understand games as feedback systems (Figure 1). With the game state representing the current condition of the elements of the system, such as the pieces positions in a chess board, the other three elements correspond to the elements of a cybernetic system's feedback loop. The scoring function is the sensor that measures an aspect of the game's state. The controller is the comparator, which makes the decisions. The game mechanical bias is the activator, an event that changes the game's states.

Positive Feedback Loop	Negative Feedback Loop
1. Destabilizes a game	Stabilizes (balance) a game
2. Rewards the winner	Forgives the loser
3. Ends a game	Prolongs a game
4. Magnifies early success	Magnifies late ones

Table 1 - Effects of positive and negative feedback loop [8].

We can observe a simple example of feedback loops in games when we consider the number of pieces of players in a game of chess. When a player loses a piece, the game's state changes, and players lose the abstract resource of strategic advantage. This loss makes it easier for a player to lose more pieces (game mechanical bias). When a feedback loop is used to amplify the difference, it is known as a positive feedback loop. Feedback loops can also be negative. In this case, the system's output is driven towards a target value, essentially creating a stable result. In the racing game Wipeout, for example, there is a negative feedback loop that makes the vehicles that are far behind the first place faster. According to [10], it is common for racing games to employ a similar mechanism because exciting moments such as fighting for the first place among a dense cluster of vehicles are more likely to occur.



Characteristic	Value	Description
Type	Positive	Amplifies the difference
	Negative	Dampens differences
Effect	Constructive	Helps a player win
	Destructive	Makes a player loose
Investment	High	Requires many resources
	Low	Requires few resources
Return	High	The net gain is high
	Low	The net gain is low
	Insufficient	The net gain is negative
Speed	Immediate	Is in effect immediately
	Fast	Takes a little time to take effect
Range	Slow	Takes a lot of time to take effect
	Short	Operates directly over a few steps
	Long	Operates indirectly over many steps
Durability	None	Works only once
	Limited	Works only over a short period of time
	Extended	Works over a long period of time
	Permanent	The effect of the feedback is permanent

Figure 2: Seven characteristics of feedback loops [3].

Marc LeBlanc reached certain conclusions regarding the behavior of positive and negative feedback loops in games shown in Table 1 [8]. Marc also points out that feedback systems can emerge from the game mechanics by accident and according to [1] they can greatly harm the system if improperly implemented. In the light of that, it becomes essential in designing mechanics to be able to identify the feedback loops and understand their impact on a game's dynamics.

Joris Dormans and Ernest Adams in [3] go deeper in the profile of a feedback loop in order to understand how it affects the system. They list seven characteristics of a feedback loop as shown in Figure 2.

2.4 Related work

There are two main works related to this paper. In the first one [5], Joris presents the Machinations framework and shows how Machinations diagrams can be used to simulate and balance games before they are built. Emergent behavior are hard to design, mainly because the mechanics of those games are hard to balance and usually designers must rely on frequent game test to achieve this balance. Joris presents a way to simulate games in early stages of development, the Machinations framework, before prototypes are built, in order to design balanced mechanics effectively and efficiently. To this end, Joris used the Machinations framework to simulate and balance the game SimWar, that has been described by Will Wright.

In [7], the authors presents the MDA (Mechanics, Dynamics and Aesthetics) framework. This framework is a formal approach to understand a game, aiming to fill the gap between game design and game development. According to the authors, this methodology clarifies and strengthens the iterative process of developers, making it easier for all parties to decompose, study and design a broad class of game designs and game artifacts. To test the framework, the authors improve the AI component of a game using the MDA, and they noticed that simple changes in the aesthetic requirements of a game will introduce mechanical changes for its AI on many levels. Using the MDA framework, the authors could reason explicitly about aesthetic goals, draw out dynamics that support those goals, and then scope the range of mechanics accordingly.

The main difference between the frameworks proposed in each paper is that the MDA framework [7] focuses on all of the design process, it goes all the way from the mechanics to the aesthetics. In contrast, the Machinations framework [5], focuses only in the design of the mechanics. The MDA does not propose a framework to model the mechanics separately, it focuses on how each component of the framework (Mechanics, Dynamics and Aesthetics) influence each other. In the end, the Machinations framework complements the MDA framework.

3 Eliosi's Hunt

Eliosi's Hunt [2] is a 3D sci-fi top-down shooter and platformer currently being developed by TDZ Games (Figure 3). The game will be released for PC, PS4 and Xbox One in the third quarter of 2016. A player plays as the young Eliosi, who dreams of becoming a bounty hunter. To complete his first contract, Eliosi will have to face monsters from nature, mutated creatures, nomad tribes, natural disasters, a robot army and others. During his journey, Eliosi can use several weapons (e.g flame-throwers, laser guns and launchers) and equipment (e.g jetpacks and thruster boots) to help him through the dangers around him. He will also be able to upgrade his companion drone that helps him in the toughest moments but in order to do that he must explore the most dangerous places in the planet.

Figure 3: Eliosi platforming in the first level.

The story of the game takes place after a war where the population of the planet had to be demilitarized. To solve the conflicts such as high criminal rates and animal raids, the people started to contract bounty hunters. The game starts at a bar where the bounty hunters usually take their contracts.

Young Eliosi is at the bar gazing at the contracts board while some of the greatest bounty hunters are choosing their contracts. Eliosi decides to take his first contract where the mission is to capture a great entrepreneur and robot builder, Sieverr, that is being wanted after banished for his creation of war machines. After his research, Eliosi finds out about a hidden factory in the middle of a swamp (Figure 4) and he decides to investigate it. In order to complete his contract, Eliosi will face 6 unique designed levels (Swamp, Factory, Volcano, Desert, Factory and City).

3.1 Eliosi's Hunt Mechanics

Eliosi's Hunt physics mechanics consist of jumping, dashing and picking up items. The jumping physics is influenced by the momentum that affects the impulse of the jump depending on the velocity of a player and the moving objects that he might be on. The impulse of the jump can also be influenced by the amount of time a player holds the jump key up to a maximum limit of time. The dash mechanic allows a player to move a fixed amount of space in a short period of time. The dash mechanic can be combined with the jump. While using dash on the ground a player becomes immune to any damage for 0.1 seconds (the time a player is in dash mode). There are 3 types of physics pick up items. The jet pack allows a player to float in the air for an amount of time. The thruster boots gives the double jump ability to a player and the speed boots let a player to run faster for a short period of time.

[2]trailer available at http://www.tdzgames.com/en#media

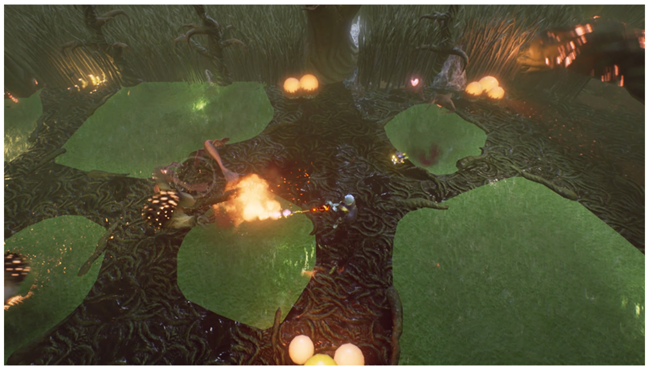


Figure 4: Eliosi in the first level - A Lady Swamp.

Progression mechanics can also be found in Eliosi's Hunt. These mechanics takes form of "gates" that prevents the progression of a player to a certain part of the map. These "gates" can be opened using specific weapons or killing the enemies in the area. Thus it gives the designer more control over how a player progress though the level. These mechanics can also be used as hidden rooms that a player can open either backtracking or figuring out a quicker way.

Another type of mechanic that can be found in Eliosi's Hunt is social interaction. The game has a mode that encourages a player to finish the levels as quickly as possible. A player's time will be displayed in an online leaderboard so that he can compare his time with his friends'.

Finally, we get to the last type of mechanic in Eliosi's Hunt, internal economy. There are several examples of internal economy in the game. In order to use the dash mechanic, a player needs to accumulate a fixed amount of dash points. When a player uses his dash, he loses all of his points from one slot. He starts the game with only one slot of dash point, but he is able to upgrade his stats in order to have up to three dash points slots. A player gains dash points at a fixed rate, but he can also gain points when he shoots enemies. Once a player has all of his dash points slots full, he stops gaining points until he decides to use the dash ability.

Another example of an internal economy in Eliosi's Hunt is the ammunition mechanic. A player can pick up weapons during the levels. A weapon comes with a fixed amount of ammo but a player can upgrade his stats so the weapons can come with more ammo. There are certain places in the level where the weapons available to pick up will come with less ammo then the normal amount, even if a player has already upgraded the ammunition stats. The weapons consume ammo in two different rates. The flamethrower, for example, consumes the ammo every frame as long as a player is holding the fire button. On the other hand, the machine gun, for example, consumes ammo per shot fired. The only weapon that has infinity ammo is Eliosi's pistol so a player can always have a weapon available for use.

Eliosi's hunt main internal economy is the energy system that Eliosi collects in order to power his drone that will protect him. This works as Eliosi's life system. A player starts with 3 available energy slots and can upgrade it up to 6 slots. All the monsters and objects that cause damage to a player have a fixed damage power. All the hits suffered by a player represents one full energy slot. Only when a player fails on a platformer challenging, like falling into a hole or lava river, he will lose all his energy slots at once. The energy that a player can collect throughout the levels usually represents less than the amount needed to complete one full slot. Only once a player complete a full energy slot that it will be used to protect him.

There are three ways that a player can collect energy. When he kills a monster he automatically collects its energy. The stronger the monster is, the more energy it gives. The other way to collect energy is destroying a check point. When a player reaches a check point he can choose to use it to save his progress or destroy it. When he destroys it, he gains a speed bonus and also all the energy needed to complete one full energy slot.

The last way to gain energy points is destroying energy objects spread throughout the levels. There are two kinds of these objects. The weak one gives less energy and the designer can determine how much energy each object can give depending on his placement on the level. On top of the amount of energy that a designer choose per object, there is a random deviation that can go from 15 percent less up to 15 percent more. There is also a death multiplier that influences the final number of energy that the object gives. The more a player dies, the bigger the number is. This works as a negative feedback loop in the game, helping a player who is having more trouble to go trough the levels. The strong energy object always gives all the energy needed to complete the next energy slot, just like when a player destroys a check point. Finally, when a player dies, he comes back to the last saved check point with the half of the collected energy he had when he saved that check point (a player always have at least one full slot of energy).

4 The Machinations Framework

Game mechanics may not be clearly visible in a game. There have been a lot of attempts to create models to describe them. The Machinations framework, described in [4], was designed to retain the structural features and dynamic behaviors of games they represent. Program code, finite state diagrams and Petri nets are examples of models that are sometimes used to describe game mechanics. These models are complex and not really accessible for non-programmers designers. Furthermore, these models lack on the level of abstraction on which feedback loops, for example, are not immediately apparent.

The Machinations framework is driven by the vision that gameplay is determined by the flow of tangible, intangible and abstract resources. These flows are represented in the Machinations diagram and they make it easier to identify the feedback structures that exist within a game system. It is these feedback structures that most of the time determine the dynamic behavior of game economies.

The Machinations framework was designed to model activity, interaction and communication between the parts of a game's internal economy. In other to accomplish that, the Machinations framework uses several types of nodes that pull, push, gather and distribute resources. Figure 5 shows the basic elements found in the Machinations framework [5].

The most basic type of node is the pool. The pool represents where resources are gathered. The flow of resources is dictated by resource connections represented as solid arrows. They can transfer resources at different rates. The time step at which the diagrams operate can be easily adjusted. In each iteration, the nodes in the diagram may fire automatically or they can be interactive. The interactive nodes represent a player actions and are activated when clicked. When a pool fires, it will try to pull resources through any inputs connected to it. A pool can also be set to push mode. In this mode, it pushes resources along its output connections [5].

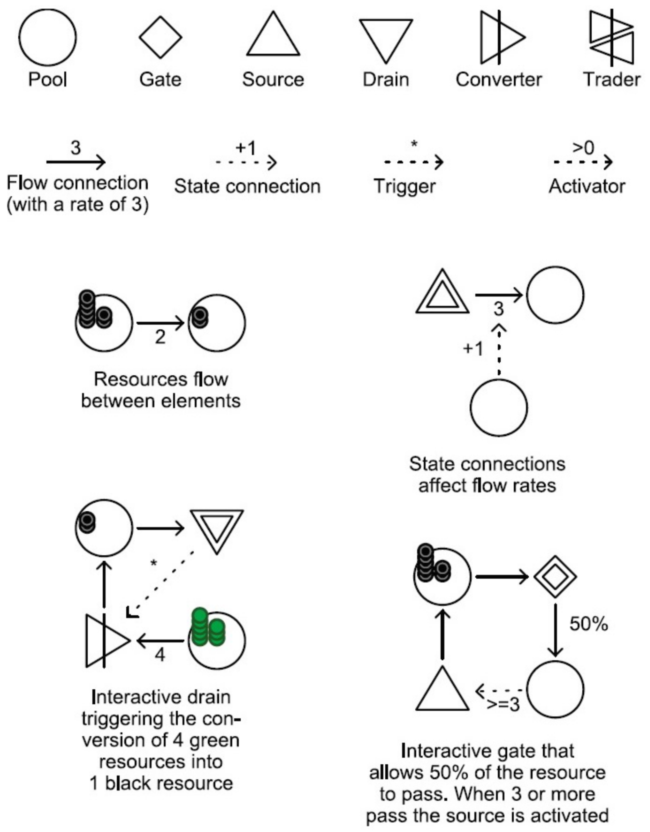

Pool
Gate
Source
Drain
Converter
Trader
3
Flow connection (with a rate of 3)
+1
State connection
*
Trigger
>0
Activator
2
Resources flow between elements
3
+1
State connections affect flow rates
*
4
Interactive drain triggering the conversion of 4 green resources into 1 black resource
50%
>=3
Interactive gate that allows 50% of the resource to pass. When 3 or more pass the source is activated

Figure 5: Elements of Machinations diagrams and example constructions [5].

When resources move from one place to another, the state of the diagram changes. It is possible to use the state changes to modify the flow rates of resource connections. Triggers are state connections that when fired, the target node will be activated either pulling or pushing depending on its connections and resources available. On the other hand, the activators are state connections that activate or inhibit their target node based on a specific condition [3].

There are other types of nodes in a Machinations diagram besides pools. In contrast to a pool, a gate is a node that does not collect resources. Its function is to redistribute resources via a probability or a condition label instead of a flow rate. There are four types of nodes that represent the basic economic functions. Sources are another type of node that creates resources. The rate of which a source produces resources is indicated by the flow rate of its outputs. Drains are nodes that consume resources. A resource that goes into a drain disappears permanently and its drain rate can be steady, random or intervals. The converters nodes convert one resource into another. Finally, the traders nodes change the ownership of the resources when fired. It is important to note that traders should be used when needed to exchange (not convert into) a number of resources of one type to another amount of resources of another type [5].

Another feature of the Machinations diagram is end conditions. It checks at each time step, if the condition is fulfilled, the diagram stops working immediately. The Machinations diagram also offers the use of artificial players to simulate players interacting with the diagram. This introduces the possibility of running multiple automated tests. Their behavior can be affected by the state of the diagram [3].

The Machination tool [3] offers the possibility of collecting data on the behavior of a game system before a game is built. It allows the designers to test different playing strategies and foresee undesirable and dominant strategies to test a game's balance [3].

5 Applying Machinations to Eliosi's Hunt

Eliosi's Hunt mainly focuses on player skill to create challenge. In the light of that, we decided to create a simplified internal economy (at least from a player's point of view) that would not require much strategic thought. Figure 6 shows the first diagram of the energy mechanic. This simplified system was first designed as a source producing a fixed amount of energy (or static engine) and a converter to turn 100 energy resources into one life resource. The static engine was created to represent the action of a player shooting items throughout the level that would generate a constant flow of energy points. The conversion of energy points into life resources is automatic (represented by the * icon) and only when a player has 100 energy points that it converts them into one life point (represented by the & icon).

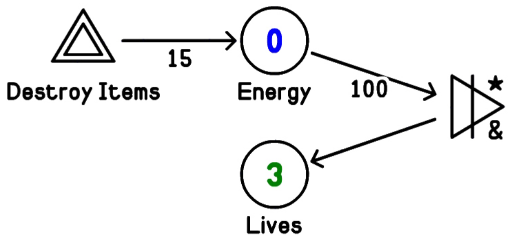


Figure 6: First diagram of the energy mechanic.

A player would accumulate life points as he progresses through the levels and destroys energy items. Figure 7 presents three new mechanisms: the option to fight enemies, the game over condition and the availability of energy items. The fighting mechanic was abstracted in this diagram in order to focus on the internal economy. It is represented by an interactive gate with a joystick icon, that means this action takes player skill into account. In order to analyze a larger number of player skills in the diagram it was used a random number (the die icon) to make sure a player would lose a life point and die in some of the fights. The game over condition is simply a test to stop the diagram when the lives pool reaches zero resources. The energy item mechanism was expanded in order to match the in game system more faithfully. Because a player can only destroy each energy item (and collect its energy) once, the energy resource source was replaced with a converter, which requires the energy items available resource. This resource is generated by a slow static engine (it generates one resource for every five time steps), representing a player's progression through the level.

This mechanic creates a problematic deadlock situation. A player has two available actions: to fight enemies (which may cost one life resource) and to destroy energy items (that are needed to indirectly generate lives). The deadlock arises when you consider that a player needs the available energy item resource to destroy energy items (and that its source depends on a player progression through the game space) and that he needs lives in order to fight. In order to solve this deadlock, a simple mechanism was created represented by Figure 8.

When a player fights and dies, he will lose a life resource but it will also restore all the available energy items (note that the pool is limited to 10 resources so that dying does not create more and more resources, creating an unwanted positive feedback loop). Although this solution helped prevent the deadlock, it also requires a player

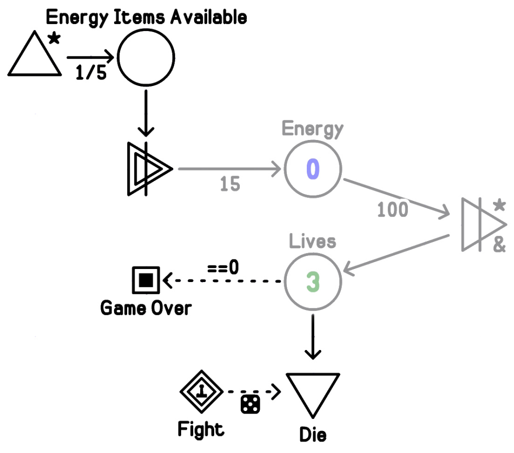


Figure 7: The three new mechanisms.

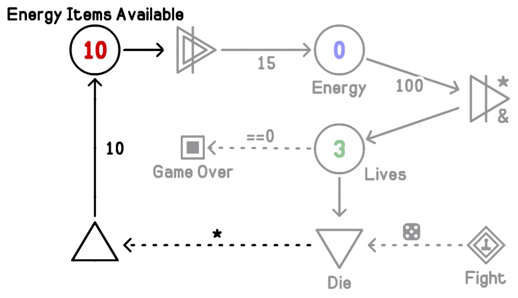


Figure 8: Solving the deadlock.

to destroy more energy items as soon as he died in order to get the life resource back. This practice is known as farming and can lead to frustration especially if it is require right after a player failed a challenge (died in a combat situation).

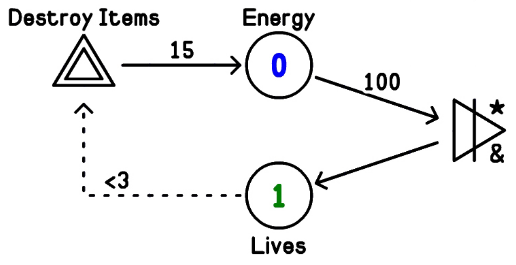


Figure 9: Creating a new basic structure.

The base structure originally proposed in order to simplify the internal economy for players was proving to be difficult to balance. Based on that fact, it was created a new basic structure for the mechanic, as shown in Figure 9. To prevent the unwanted incentive to farm resources through death, it was created a maximum amount of life resources.

With this new approach, the game over condition was not needed anymore, since the fight could trigger a life loss, instead of a game over condition. This is represented by Figure 10. The new fighting punishment is lighter than before, which enabled the game to push a player's skill further. Now, when the life resources reaches zero, it simply triggers a source and get it back to one. Although this new system made farming less needed, it was still implicitly encouraged

[3]available at www.joridormans.nl/machinations

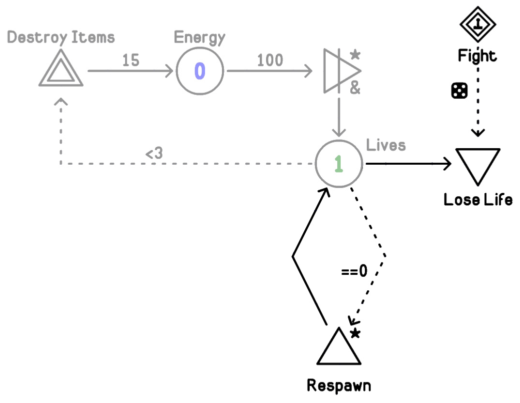


Figure 10: The new game over condition.

if a player failed the same challenge many times. In this case, he would have to destroy more items than before in order to try the same challenge.

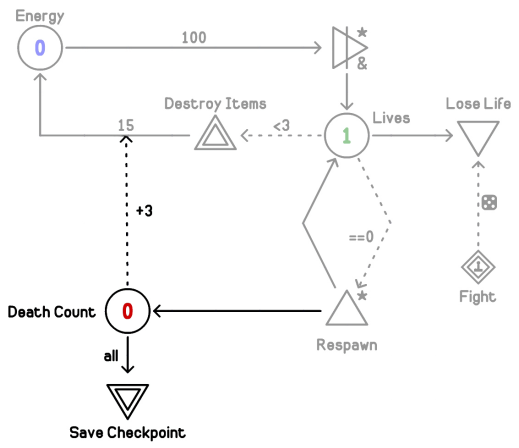


Figure 11: Adding a negative constructive feedback loop.

To solve this problem, we created a negative constructive feedback loop, as shown in Figure 11. This feedback loop is triggered by a player's death and causes the number of energy points per destroyed item to increase. Although the return of the feedback loop was low and its range was long, it was needed a mechanism to prevent a player from stacking up deaths counts and making the feedback loop permanent. It was included in the checkpoint mechanic a simple drain to remove all the death count resources when it was activated. In order to prevent this diagram from getting too confusing, we decided to abstract the whole checkpoint mechanic into one interactive drain, keeping the focus on the energy mechanic.

This feedback loop was great to help players that were dying often to get through the challenge without needing to farm energy points. This, however, is a type of Dynamic Difficulty Adjustment (DDA) that is, the game adapting itself based on player performance in order to present the right amount of difficulty for that player's specific skills. Although DDA is a great tool for designers to make sure all types of players will enjoy the game, as Jesse Schell points out in [9], there are several problems associated with it. For one, players may find that it spoils the reality of the world, they may exploit the system or they may be outright insulted by this system.

In order to avoid these problems, we changed the amount of energy gained per energy item destroyed to a random value. According to [3], random values in internal economies makes it difficult for players to assess the strength of the feedback loop. Figure 12 represents these changes. There is another problem associated with this feedback loop: players who upgrade their avatar skills (increasing the maximum number of possible life resources or decreasing the amount of energy needed to be converted) may quickly reach a high number of life points. To counter that, we created a positive destructive feedback loop, also shown in Figure 12. The feedback loop causes the number of energy points per destroyed item to decrease the more life points a player has. This makes sure that players will not stack up a large number of life points and make the game easier. These two important feedback loops balance themselves and creates a promising scenario for emergent behavior to occur, because according to Jochen Fromm in [6], systems with multiple feedback loops display more emergent behavior than systems with only one.

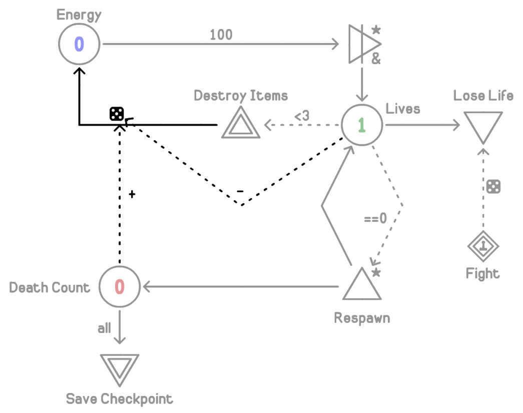


Figure 12: Adding randomness to the mechanic and the new feedback loop.

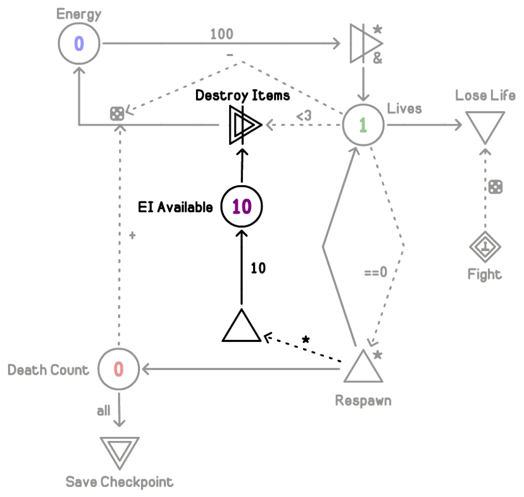


Figure 13: Elaborating on the destroy item mechanism.

As with the first base model, we elaborated on the destroy item mechanism in order to represent more faithfully the dynamic of the game. In Figure 13, the source labeled "destroy items" is replaced with a converter that takes in the EI Available resource (amount of available energy items) and creates a random number of resources that is influenced by the two feedback loops. Just like the previous system, there is a source that refills the pool to its maximum (10 resources) when a player dies. However, because in this new system the respawn action will occur less often than in the previous one,

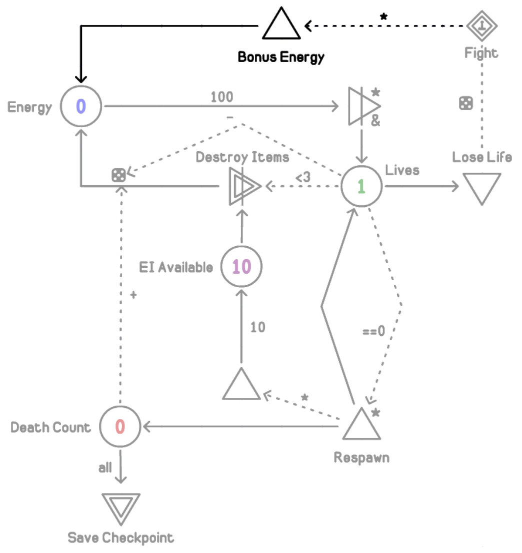


Figure 14: Creating a new energy source.

this situation still contains a deadlock: the number of energy items available will quickly run out, and a players' only option would be fight until there is no more lives and respawn.

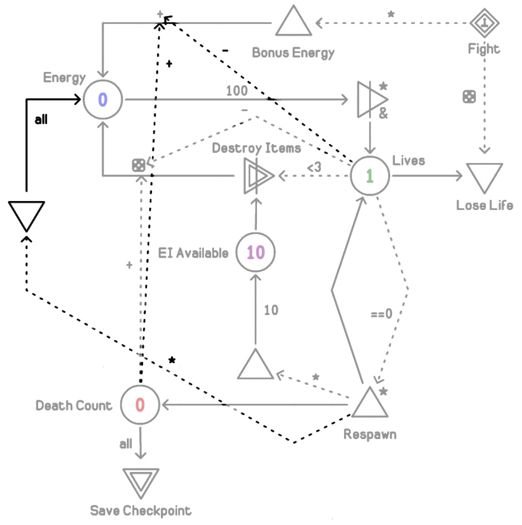


Figure 15: Balancing the constructive feedback loops.

In order to prevent this deadlock, another energy source was created. As shown in Figure 14, when players fight they will now activate a bonus energy source that creates resources for the energy pool. In the complete fight mechanic, this source would be triggered by a player killing a monster and the amount of energy generated would depend on the strength of the monster. The new source and connections represent another feedback loop, positive and constructive: The more energy (that would turn into lives) that a player has, the more he will fight, which will give him more energy and so on.

In order to keep the new feedback loop balanced, the two previous ones (or at least the destructive one) should also act on the amount of energy generated by the source. Figure 15 represents these changes. Another mechanism was created to balance the new constructive feedback loop: when a player respawns, the energy pool is emptied (also represented in Figure 15).

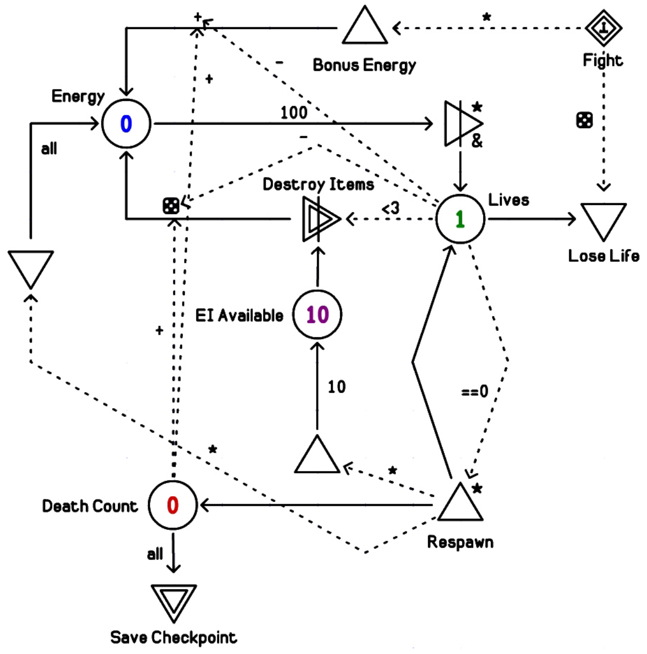


Figure 16: The complete diagram of the energy mechanic.

In Figure 16, it is presented the entire energy mechanic of Eliosi's Hunt. Note that there are other mechanics abstracted in this diagram, such as the checkpoint and the fight mechanics in order to keep the diagram understandable. With this holistic view, we can properly analyze and understand the needs of the mechanic, such as two deadlock situations that emerged and we can act to improve the mechanic, like adding the random element to the first two feedback loops in order to make the DDA less apparent for players. Visualizing feedback loops in the diagram makes it simple to understand its profile and easy to balance the game. For instance, when we added that last feedback loop (which was constructive) there was a clear need to add new mechanisms to balance the now easier mechanic.

6 Conclusion

The game mechanics design process is difficult for designers to iterate on and balance with the current widespread design methods. The Machinations Framework was designed by Joris Dormans as a way to allow designers a complete and simple view on their systems. This paper proposed the application of this framework to model and design Eliosi's Hunt energy mechanic.

By using the Machinations Framework, it was possible to quickly identify deadlocks and iterate on the solution. Without a formal mechanics design method, it is difficult to identify and create complex systems involving feedback loops, because they get obscured in the game code itself. The framework allowed us to identify the need of a feedback loop and its characteristics, as well as how they should interact with one another in order to create balanced mechanisms.

The Machinations Framework only models the mechanics of a game, focusing on the internal economy. In the light of that, the main approach for future work is the study of tools that helps a designer to create the dynamics and the aesthetics of a game and also the other types of mechanics that the Machinations does not model very well, such as physics, tactical maneuvering and social interaction mechanics. After the study of those frameworks, it is important to use them to model real games, and test how much it helped the design of a game.

References

[1] E. Adams. Designer's notebook: Positive feedback. Gamasutra 2002. Available online at: http://www.gamasutra.com/view/feature/131426/designers_notebook_positive_.php.

[2] E. Adams. *Fundamentals of Game Design (3rd Edition)*. New Riders, 2013.

[3] E. Adams and J. Dormans. *Game Mechanics: Advanced Game Design (Voices That Matter)*. New Riders, 2012.

[4] J. Dormans. Engineering emergence: Applied theory for game design. PhD thesis, University of Amsterdam, 2012.

[5] J. Dormans. Simulating mechanics to study emergence in games. In *Proceedings of AI and Interactive Digital Entertainment Conference, Palo Alto CA*, October 2011.

[6] J. Fromm. Types and forms of emergence, June 2005.

[7] R. Hunicke, M. Leblanc, and R. Zubek. Mda: A formal approach to game design and game research. In *In Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence*, pages 1–5. Press, 2004.

[8] M. Leblanc. Feedback systems and the dramatic structure of competition. GDC 1999. Available online at: http://algorithmancy.8kindsoffun.com/cgdc99.ppt.

[9] J. Schell. *The Art of Game Design: A Book of Lenses, Second Edition*. A K Peters/CRC Press, 2014.

[10] K. S. Tekinbas and E. Zimmerman. *Rules of Play: Game Design Fundamentals (MIT Press)*. The MIT Press, 2003.

[11] N. Wiener. *Cybernetics, Second Edition: or the Control and Communication in the Animal and the Machine*. The MIT Press, 1965.