VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ADAPTIVE PLOT GENERATION IN ROLE-PLAYING GAME

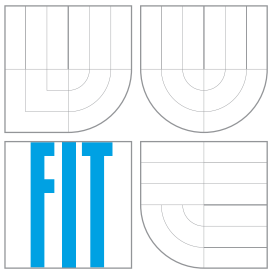
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ VYMAZAL

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ADAPTIVNÍ GENEROVÁNÍ PŘÍBĚHU V RPG

ADAPTIVE PLOT GENERATION IN ROLE-PLAYING GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ VYMAZAL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK HRBÁČEK

BRNO 2015

Abstrakt

Procedurální generování příběhu nabízí mnoho výhod například pro přostředí počítačových her. Ovšem, zachování alespoň některých kvalit autorského vyprávění při algoritmickém generování je náročný problém. Tento je v práci diskutován současně s přehledem existujících přístupů a řešení. Dále je v práci prezentován návrh řešení založeného na evolučním počítání a výhody tohoto přístupu. Následuje popis použití tohoto přístupu na tvorbu a řízení jednoduchého příběhu v RPG prostředí, vyhodnocení výsledků a srovnání se současnými metodami.

Abstract

Generating a story, while trying to preserve at least some qualities of author-written narrative is a complex issue. In this thesis several currently existing systems and approaches are discussed. Then, a solution based on evolutionary computation is presented, and its traits shown on small-scale proof-of-concept scenario. Finally, this approach is compared againist existing solutions.

Klíčová slova

Evoluční počítání, Interaktivní příběh, umělá inteligence, počítačové hry

Keywords

Evolutionary computaion, Interactive narrative, artificial intelligence, computer games

Citace

Jiří Vymazal: Adaptive Plot Generation in Role-Playing Game, bakalářská práce, Brno, FIT VUT v Brně, 2015

Adaptive Plot Generation in Role-Playing Game

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Inženýra Radka Hrbáčka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Vymazal
May 20, 2015

Poděkování

I would like to express my sincere gratitude to Ing. Radek Hrbáček for having patience with me, his valuable advice and guidance of this work.

Contents

Chapter 1

Introduction

In modern computer games, we can see a tendency for creating ever more immersive experience for player. This often, especially in some genres, such as role-playing games, cannot be achieved without compelling story. There are multiple ways how to try to create such a thing. We will go over some of them, some cases in greater detail than others.

Evolutionary computation is a very interesting field of artificial intelligence and has numerous possible usages. Most of its applications are designed and deployed only of late, because in past computational requirements of these methods were rendering them hard-to-use. We will look into how this approach can help us with the interactive narrative.

At first we should answer a question why, or whether we really need interactive narrative. Well, of course there are games where we hardly need any story at all, such as simulations or puzzles. But even across those genres nowadays a trend can be clearly seen, game developers are trying to get some sort of story into them, however simple and short. So need to have some reason for so big effort putting a story where some could says no story belongs.

Beside what is proven by numbers – a game with some integrated, even pseudo-story are simply more positively accepted, is more selling, downloaded etc. In general this is caused by player being more attached to the game, human mind helps us here, for being able to integrate with lot of things. This is what is able to create stronger mindlessly-click-or-fail urge even in a very simple game.

However, this was not interactive narrative, for that we need to move to more sophisticated genres, such as role-playing games, where we will stay. Here, on the other hand, very simple story with linear progression quickly begin to be boring, because the game is much more story-oriented. If player is to have the option to really live trough the narrative, possibilities like perceived freedom (4.2) and ability to alter the story trough action are needed. Original pre-authored narrative can give us this sensation in case it is so compelling that it keep us going in desired direction, thus swaying us from trying and pushing the system somewhere, where its limitations would be apparent. Even though, this feeling last only for first time, so it offers little to no replayability.

In addition, such narrative is not scalable and must be designed for exact setting, which seriously hinders any reusability even when designing the game, creating necessity to go through this time-consuming process again and again. And even if you accept this, it is not possible to make a story so each person ever playing it is happy with how it is designed, because we all have different personalities.

Above mentioned problems are among those which interactive narrative is trying to overcome, without hard-set story to go along, you are able to adapt to needs and preferences of player, different setting and change it every time, thus gaining so valuable replayability.

What more if you manage to design it as more less general system, you can use it to repetitive fast-design of new plots and games.

In chapter 2 we look into some possible approaches to interactive narrative and discuss their advantages as well as not-so-good aspects. In chapter 4 we will move to describe in detail how evolutionary computation can be used to address this issue. Next, in chapter 5 is presented example of small-scale game utilising aforementioned design. Chapter 6 compares this approach with those listed in 2. Chapter 7 concludes this thesis with summarization and outline of further development possibilities.

Chapter 2

Existing solutions and systems

Idea to make computer create and interactively shape stories for us is not new, although computational power needed for more complex artificial intelligence is generally available only lately. Early works like [6] have been addressing this and first systems were created. Later expanded by [5] foundations for some of the systems discussed below were laid. This is of course far from complete list, but it is sufficient for brief insight into problematic and for context evaluation of evolutionary-based solution later.

2.1 The Yawgh

One of the options, and widely used one, which is why we have here example of actual finished and sold game, is to create pre-authored narrative so branched and diverse, that player is not aware he is actually just picking one of pre-defined options, which create very sound illusion of truly interactive story, while preserving most of pre-authored narrative qualities. In case of The Yawhg[14] this is achieved by designing not one, but four (game is intended for four players) narratives, which are mutually intertwined. Whole game consist of one week, that is six day lasting stages, where each player gets to make one action and conclusion on seventh day. By this, we get four several-valued vectors which are mapped into results space and ending presented to players, everything is deterministic, no matter how it might look on the outside. Thanks to this huge operating space and completely controlled output, results often link to player actions in very intrigued ways. However after three to four playtroughs (which are short, game consist solely from players choosing their daily action, possibly with several-clicks decision tree) player starts to quite see how their choice and output is linked which leads to gradual loss of interest for, now completely predictable, story development.

2.2 Mimesis

An initiative to create abstract system with API designed for general usage with any conventional game engine[15]. The system takes direct control over game world and entities and employs resolver to compute action sequencing of activities available in the game engine (original idea of STRIPS system [6]). These are afterwards deployed and their execution controlled, and adapted in case of not going according to plan. System is designed for classical 3D environments and is focused more on creating detailed plans (including camera and similar game aspects) for controlling some very low-level action in detail, rather than

creating narrative for game as whole. Implementation includes ways to deal with player acting out-of-plan both by performing masked intervention (limiting player in a way that does not collide with perceived freedom) and story re-planning, which is however done only after pre-calculating that it requires only small plan correction. While true interactive narrative, system is still fully deterministic, requiring solid frame which it fills with actions. Mimesis API is implemented as C++ libraries.

2.3 GADIN

Another, bit newer, STRIPS[6] based approach, GADIN stands for Generating Adaptive Dilemma-based Interactive Narratives[2]. GADIN takes STRIPS condition-satisfying approach and expands it by introducing dilemmas, in whose player has to take side and thus directing the story. This is achieved by affiliation system between player(s) and/or NPCs. Opposed to Mimesis system mentioned above(2.2), GADIN is text-based and not bound to any engine or specific environment. It also handles main story goal selection and directing story to it, switching main goal if found unachievable. This, combined with fact that we are in purely virtual environment (not a graphically represented one) allows for higher degree of freedom and less restrictions on user actions.

2.4 Character Involvement

This approach is different from all above mentioned in the fact that it facilitates partly stochastic approach. It builds upon concept of *hierarchical task networks*[5] and, in addition to story planning by satisfying conditions, introduces character reasoning and intents, producing more believable narratives. This means that system has to keep balance of goals of whole story with goals of individual participants. Specifically, this solution uses Intent-Driven Partial Order Causal Link (IPOCL) planner, which contrary to STRIPS[6] and STRIPS-like systems uses iterative and only partly deterministic approach. It is considered in [11] where is elaborated how considering bigger-frame character intent helps narrative and given empirical evidence thereof.

Chapter 3

Background

As we have seen in previous chapter, there are many interactive narrative systems and approaches in existence. Yet a *specific design* solution based on evolutionary computation has never been published. While there has been usage of evolutionary approach, for example in game-AI design[1] and level content generation[8][13], there is not case where it would be used for generation of interactive narrative as whole. Now we will look at evolutionary computation in general, in next chapter we will look on the possibility and the properties of a concrete design of interactive narrative.

3.1 Evolutionary Computation

Nice overview of this topic can be found in [12]. Following is brief introduction with respect to consecutive design of interactive narrative system.

Evolutionary computation, or evolutionary computing is branch of artificial intelligence utilising nature-inspired approaches to solving a problem. Its foundations lie in work of C.Darwin and is inspired by principles of natural selection, that is, the survival of the fittest. Further, I will focus on evolutionary algorithm, from now on referred to as EA. An evolutionary algorithm is, in general, a population-based, meta-heuristic optimisation algorithm. It can use operators such as recombination and mutation, which are further defined and specified by individual representation.

One of the key elements for our case is that evolutionary computation techniques often exhibit the property called emergence. It can be (and throughout the course of history was) described in many ways, for example, we might say that emergence occurs when a complex system gives rise to a perceived pattern at a higher level than the rules of the system operate at. The higher-level system operates with its own rules and patterns that are implicit in the definition of the lower-level system, yet are not explicitly modelled in the system, and can thus sometimes give rise to unpredictable behaviour. Perhaps most apparent manifestation of this in evolutionary computation field can be seen in cellular automata. The most generally known cellular automaton is The Game of Life[7] which employs a small set of very simple rules, but is able to create complex patterns displaying behaviour far beyond complexity of and scope of those rules. In image 3.1 we can see breeder itself (in red), next path of glider-guns (green) it leaves as it is moving and small gliders produced by the guns (blue). Such complicated behaviour is far beyond scope of simple 3-rule sytem, where each rule concerns only number of immediate neighbours to one cell.

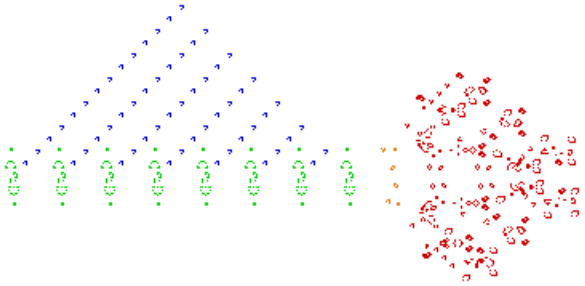


Figure 3.1: “Breeder” from Game of Life

3.2 Evolutionary Algorithm

In principle, EA goes as follows: begins with creating initial population, either randomly or using domain-specific knowledge. Number of individuals in population can be hard-set or flexible (random or heuristic). Then main loop is entered, which is exited upon satisfying ending condition, or running out of time. In each pass of the loop, called generation, first step is to choose parents (if said algorithm features recombination). Parents can be chosen by fitness (deterministically), by tournament selection, randomly or any combination thereof. Then new individuals are created, generally by recombination and/or mutation method. All the new individuals are evaluated. Next, individuals for new population are selected, the selection can be made from new individuals created in preceding step, parents, or even older generation. There are many approaches to selection, featuring different share of deterministic and stochastic methods, however generally, individuals with higher fitness have higher chance of being in next population.

EA belongs to a class of solution-space searching algorithms, and as such face fundamental problem of balancing two seemingly different goals. It has to thoroughly search surroundings of best solution currently available while still venturing far enough not to get stuck in some local optimum. This dictates that each EA has to be finely attuned to specific solution-space and discards possibility of universal algorithm for wide set of problems.

3.2.1 Evolutionary Programming

Our problem is, concerning EA paradigms, not particularly easy to classify, mainly because of how individual is represented (see 5.4). However, of the main branches of research, it is closest to *evolutionary programming*[3] which poses no limitation on individual representation, leaving it to completely domain-specific solution. Also, like my solution, it typically does not use any recombination mechanism, leaving evolution to mutation operators. Evolutionary programming shares some traits with evolutionary strategy, such as not using crossover operator and carrying out same evolution steps on each individual from parent population. Notable difference is that evolution strategy more often selects offspring deterministically by fitness, while evolutionary programming introduces at least certain level of stochasticity.

Chapter 4

Design of Evolutionary Computation-based Solution

From the properties of evolutionary algorithms discussed in previous chapter, potential for their usage in the interactive narrative can be seen. In this chapter we will look upon how to exploit it and turn into game and narrative generation design.

4.1 The Game

First, to somewhat classify game being design, what I aim to do is to create digital version of tabletop role-playing game. For example we may take the most widely-known one, *Dungeons and Dragons*[10] (further referred to as D&D). D&D utilises highly evolved and very complex system designed to allow DM[1], framework for creating sufficiently believable story and to provide for all players enough options as to what to do in game and how to achieve goals. In our case I will use much more simple system inspired by D&D one. First-of-all, it is designed for only one player and DM, program takes role of a DM. In addition the rule-system will be much simplified, utilising fact that we are in digital space. This will help us to solve few problems by more convenient way than one used in official pen-and-paper solution. I will retain the scheme that player controls only the actions of his character in game and DM, here supplemented by the system, controls everything else. Next, player input will be not spoken one but textual instead, which will be only way player can interact with game. Game will "speak" to the player trough both textual output and trough GUI visualisation (for some things like map, character statistics etc.). In example application I will use classical high-fantasy RPG setting, such as one most commonly used in D&D, but developed storytelling system is general and can be easily used in any setting (more on that in 4.3).

Narrative game, that is not one telling the narrative, but one creating it, or, to be more precise co-creating it with player trough his actions is so complex issue partly because of problems of context. This covers how to tell where information given by player belongs. There has been, and still continues, extensive theoretical work covering this issue, for our case I will briefly touch this work [4]. I will not go into great detail here, because the design

[1] Dungeon-Master, sometimes called also Game-Master or similar terms across different games

of game serves mostly to overcome this problem by elimination of all but one frame from actual text interaction.

By having game completely single-player we lose some of the context complexity because there is no dispute over to whom conveyed information is directed to. Also, as DM is a program, there is no discussion with the player as there might be when they are human. Story is generated without possibility of *direct* player intervention, so that is another context frame we need not to care about. By limitations of text parsing and recognition, it will currently not be possible for player to directly speak to NPCs, use of commands will be required to perform desired interaction. In addition to that, by use of commands, however can they be likened to natural sentences, game can clearly distinguish what is meant and pointed at.

4.2 Communication with Player

As noted in opening of this chapter, only available form of player input will be that of simple text. Of course, to alleviate player from burden of typing long or difficult names, GUI may write name of object which player clicks on or marks in some way to the input line. This way we may keep simple communication model without being unfriendly to the player.

First, we need to split incoming string to words, determine which of them denotes action, and find corresponding command. Then we look into commands table on arguments identified command takes and we search rest of words for matches with these parameters. For this, parser can use queries to game engine for, for example nearby game entities. Another thing we need to consider are questions. If "?" character is detected at end of string we need to take another approach. Then, usually again by query to game engine, object of question is identified, and information about it available to player at that time is put to string and sent to output, without invoking game action. If pronoun is detected then special, implicit, argument is added to command for which game engine will substitute last used compatible target. This might seem dangerous action to take, but player is not expected to use it unless it is really obvious what is meant by it. Filling words, adjectives and similar elements are discarded. If command is recognised and arguments filled, it is passed to game engine for processing.

When engine receives command, first it needs to verify its syntax and availability. Syntactical analysis consist of determining if the command has available target, for example command to attack must have at least one viable target (NPC, object etc.) or there must be exactly one compatible target for given command, for example only one item in vicinity to pick up. If command passes syntax check, semantic validity is evaluated, that is if player is allowed (by rules of game world) to execute such action. This can result in success, failure, or probability of success, in which case dice is rolled. If everything went well, command is now processed and response from game collected. This information, or problem description if one of above steps failed is then conveyed to player by text output or one of visualisation elements of GUI. Whole process is illustrated in image 4.1.

4.3 Generating the Story

Now we have a system for communicating with player and we can move to building the narrative. In my solution I aim for maximum flexibility and replayability of a story, so I am not using any predefined structure. Instead, a number of small, simple event chains will be

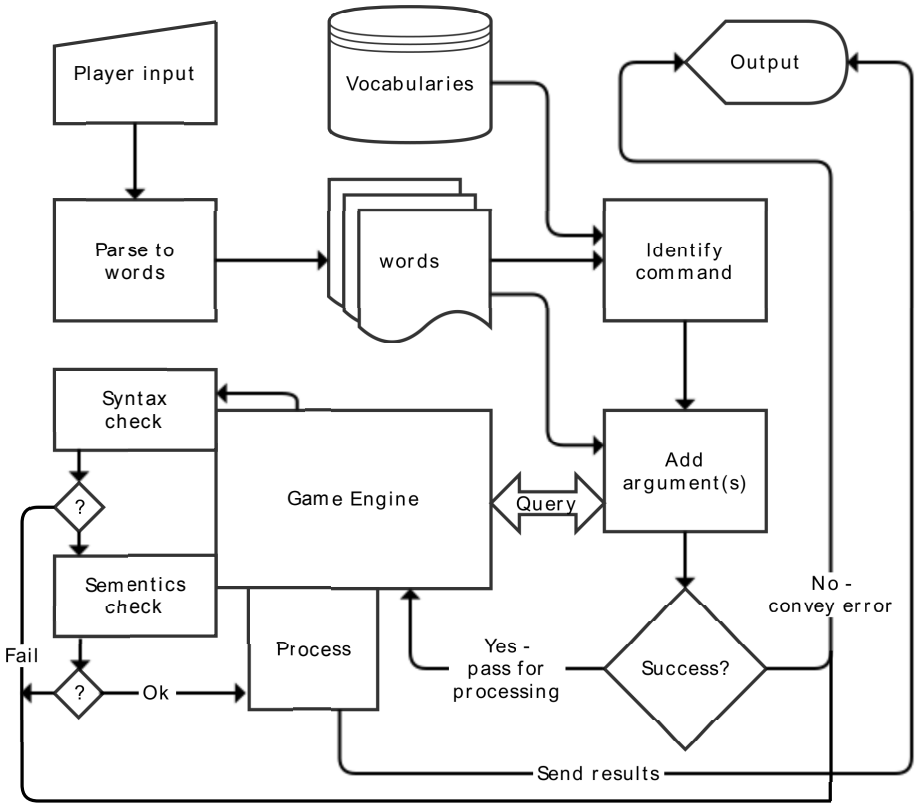


Figure 4.1: diagram of player communication loop

employed and EA will be used to construct the story from them.

To better see the idea behind this design, we can take any narrative at least close to one used in role-playing games, say, a fairy tale, and, if story is more complex, dissect it to smaller parts. If we take those to simplified, abstract level we may see that usually such a story consists of hero having to do something, that is some goal or achievement for him. This is almost always connected with some complication, someone or something which is in some state that collides with hero succeeding. So hero must change state of the complication to one which will allow him to continue. Often, this is supplemented by need to protect or provide for someone or something, which can be expressed as doing or not doing actions preventing the important someone or something from changing its state to one not desirable.

4.3.1 Story Representation

From this we may start creating interactive narrative system. I use a concept of *narrative resources* to ensure that story can be constructed from completely independent blocks, but still kept coherence. Narrative resource is abstract object, which has certain type (NPC, item etc.). This resource is than mapped to actual game entity, to allow for player interaction. Another, perhaps most important property is the status. This is the value describing resource condition relative to player. This property has huge advantage in fact that it can retain its meaning even for distinct resource types. Story in my solution then

consist of different resources state changes, either as consequence of player action or as scripted event. Resource will also have memory of which states it has gone through, which can help us as one of deciding factors when evaluating the story.

Now I can introduce the basic element in more detail, from now on I will refer to it as story *node*. Foremost, node will need some resources, with possible requirements on their status, which will determine if these can be generated (new to game world) or has to come from another node and in which condition. One such node will then feature a hook, that is something which catches player's attention (or not, that is up to player). Then there will be some objective, which can consist of more needed state changes, but node should be kept short and simple. Objective will be accompanied by some obstacle which has to be removed or otherwise overcame, to complete the objective. Finally there is some sort of reward for player to motivate him to complete objective. This reward may or may not be mentioned to him prior to his acceptance or completion of objective. Ideally there should be more ways of completing objective or overcoming obstacle, but this is by no means saying that every possible action which player can execute shall be anticipated, on the other hand, system is designed to cope with player acting out of every set way and has means to adapt to it. There should be listed only few most intuitive actions with their consequences in way of changing resource states.

When we have defined resources and node, we can start thinking about putting story together, and how EA will help in this matter. For narrative to hold together, we need to have node chains there which will be created by several consecutive nodes using same resource and changing its state. This will create story arc of this resource relationship with player, which causes player to become attached to it, resulting in immersion and agency (compellingness criteria, overviewed in chapter 1 of [2]). Next, we need some story conclusion which will be provided by special node with more specific resources requirement, typically resources which already gone trough some player interaction. In this node there will be more complex scripting because it concludes the game, which means we do not need to care about how system will cope with what we leave behind but at the same time imposing requirement for satisfying ending (interestingness, another compellingness criteria).

Finally, we have reached, at least to some extent, rigorous requirements for narrative. So, typical individual representing story will be set of nodes employing resources in a way that one of expected changes of resource state on node output matches resource requirement on different node input. As node typically operates on more than one resource, this has potential to yield narrative where stories divide and meet again and different story arcs are intertwined resulting in climax provided by final node, which utilises some of resources more frequently appearing trough the narrative. This means that for evolution we will need at least three distinct kinds of mutation operator: one for adding a node, second for removing one and third for adding the ending.

4.3.2 Evolutionary Algorithm

To ensure that something can always be built there is requirement that there exist at least some nodes which have requirements allowing to generate all needed resources. This fact also conveniently solves problem of starting the evolution, we simply start with empty individual and let EA build narrative literally from scratch. Because resources state changes are invoked in large part by player, we are limited to population consisting of single individual. This is used relatively often in both evolution strategy and evolution programming[3]. As for next step of algorithm we can mutate individual by adding node for which we either

already have available resources (of correct type and in correct state) or there is a chance of having them soon because they are listed in another existing node output options. Or we can remove some node for which resource requirements could not be satisfied for big number of generations, or which player neglected to pursue. This can help to free resources for use in another node. Finally, last mutation is to try to satisfy final node requirements which would conclude the game, for this mutation, however another requirement, concerning player progress would be set to ensure game does not end too soon. Trough applying small number of these mutations on each of numerous copies of original individual we create new population and compute fitness for each individual.

Fitness will be based on utilisation of resources, longer possible node chains (higher percentage of used resources in advanced states versus newly generated ones), possibilities of narrative expansion (coverage of possible current nodes outputs with another nodes inputs) and script diversity (penalisation of individuals using one type of script many times). Number of generations run will be determined by time consumed by last player action, which will add to reality of developing of game world, after that, control is given back to player to change some resources state and thus allowing next development. Final ending condition is of course successful placement of final node which allows player to finish the game.

This concept also allows player indirectly directing the story, because by pursuing nodes associated with certain resources player increases state changes counter on them, which in turns gives better fitness to solution having more nodes with these resources and thus giving player even more chances to interact with them. In addition this makes system exceptionally robust against player actions such as destroying critical narrative resource which is obstacle most systems are forced to cope with by not allowing player such action (2.2). Here, nodes using destroyed resource are just deleted, by cascade constrain check few other ones that depended directly on them and in few generations new nodes are added which utilise resources freed by the deletions.

4.4 Using the Story

In previous section I introduced EA which gives us adaptive structure of nodes depicting small story parts, which reacts on player changing state of resources. Next issue is how to allow player such interaction and how to project node structure evolvement into the game world.

As mentioned earlier, we start evolution from scratch, so first we allow short time for first couple of nodes to evolve and than we, for first time, pause evolution. Upon each evolution block halt, there will be check whether there are new nodes which have available resources (not waiting for any preceding node to be completed or cut off) and have never been executed. If such nodes are found, associated script is run for first time. At this stage, script, if applicable, generates game entities corresponding to evolution resources, introduces quest hook to player (if containing explicit one), and sets triggers for its next execution. Apart from possibility of script-defined triggers, each script is invoked from game engine upon player anyhow interacting with any game entity or location associated with node or resources it uses. This way is ensured that every time player does something potentially important for node, script is run and game can react accordingly, for example, change state of resource, finish or terminate node, reward player etc.

For story credibility, concept of time is also vital. If, for example, we provide player with some task where real hurry is needed, we cannot ignore how long it takes before player executes action changing state of resources to one where task would be completed. In order

to provide for this, each action available to player has its time cost. In addition to that each action has also energy cost, which forces player to rest rom time to time. And this time is also what we use in directing evolution, that is amount of generations evolved is proportional to game time elapsed. This helps believability also in another way, which is that the node removal operator chooses node to remove, excluding those in which player is currently actively working, proportionally by their time of existence, measured again by time elapsed from player actions. This gives game world sort of more living feel. As there is no central authority such as director or story agent, collective consciousness and similar phenomena used in other storytelling systems, time is only thing left, player can be sure of. This, of course, is double-edged, can have numerous negative consequences on story quality and is one of things that should be thought about in next research and development of system.

Chapter 5

Solution Realisation

To demonstrate design presented in previous chapter I created a small sample application. Game *Dungeon_ Terminal* is implemented in C++ and makes use of Qt toolkit, in version 5.4. Use of object-oriented approach is very helpful in design of genotype (evolution nodes structure) and phenotype (game world entities) mapping and also in other features.

5.1 Graphical User Interface

For user convenience, Qt-widget-based GUI was developed and implemented for application. One part of GUI is main application window, where actual text-based communication takes place and most dynamic variables such as player health and energy levels are displayed. Next there are visualisation panels which helps to convey certain information to player, thus reducing need for out of story dialogue and avoiding context recognition issues. On image 5.1 is screenshot of main window where main area for textual output is, with input line below it. In bottom there is player name, current level, health and energy bars. On the right side there there is menu panel for opening different visualisation windows. Next, I will briefly describe implemented ones.

First, and topmost in menu, is character panel. It consist of three tabs: equipment, attributes and skills. In equipment tab there is clickable list of slots where equipment can be put, details about equipped item display in pane in right side upon clicking on appropriate slot. Attributes tab shows list of player attributes with their values. Last tab shows all available combat skills and level of player proficiency in each of them.

Next, second from top, sits inventory window. Here is simple list of items possessed by player with panel which displays additional information about item clicked at. Currently items are mostly random-generated equipment so this information may be any kind of bonus game engine is able to provide and amy belong to prettemuch any slot.

Last implemented visualisation helper is the map. It has three levels of detail, on farthest one pixel equals to one sector of game world, besides terrain, only thing displayed is player, in a way that pixel corresponding to player-occupied sector is black. On mid-detail level there are visible special locations and player does not occupy full sector now. On third, detail level, in addition to player and locations, NPCs and game-objects are also visible, support for showing entities according to player knowledge about them is not supported, only never-visited areas are covered by fog-of-war, once player visits area it continuous knowledge of what is or is not there.

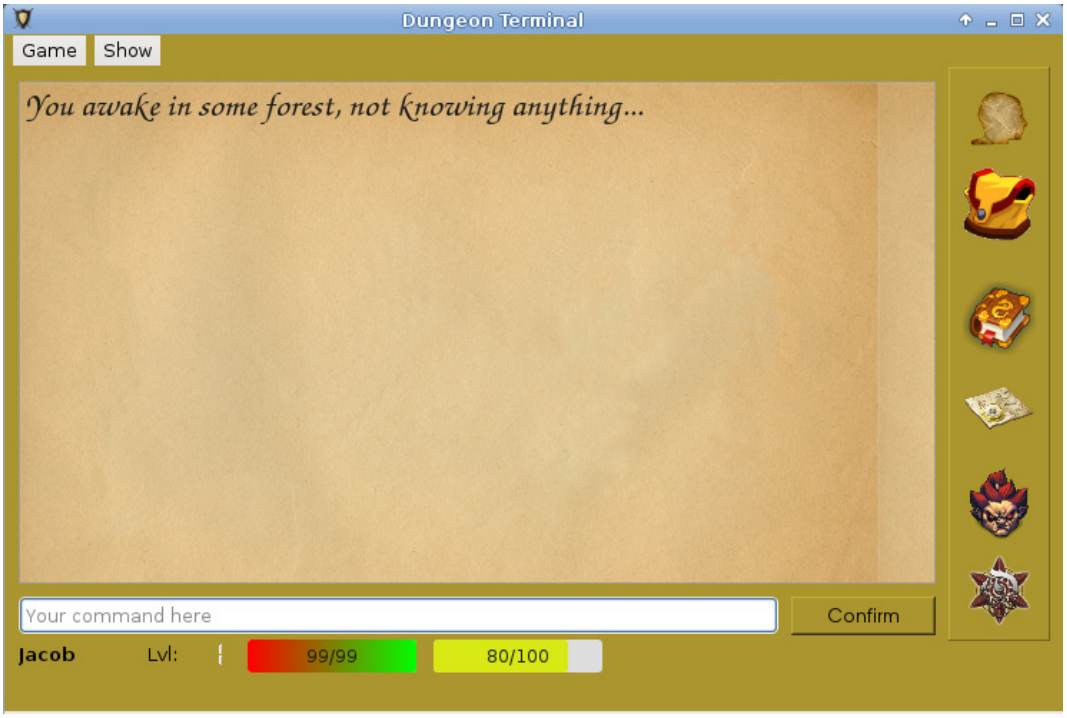


Figure 5.1: Screenshot of application main window

5.2 User Communication

Implemented parser and command processor is far simpler than one presented in 4.2. It recognises only twelve commands with zero, one or two parameters, from which one may be optional. Argument can be string or number. Command word must be entered first in input string, followed by parameters of correct type in defined order. If these conditions are satisfied command object is passed to the game engine. Else error describing what happened is sent to output. In case of optional argument there is either hard-set implicit target, such as slot with which is item compatible, or ground; or game engine searches current tile for single available target. If it fails, error is reported to player with encouragement to specify optional argument. Commands with description and syntax are listed at built-in help screen.

5.3 Simulation

Upon receiving syntactically correct command, game engine first performs checks if this is valid request and such action is in boundaries of game world available, namely if there exists valid target for targeting commands. If unable to perform action, descriptive error message is sent to output instead, explaining, if possible why requested action cannot be performed.

If rules requirements are satisfied, game engine executes the command. This can lead to numerous effects. Firstly, if any game entity is anyhow involved in command processing (in most cases there would be at least one), evolution engine is invoked and given id of this entity. Then search is done whether this game entity corresponds to any narrative resources. For each such resource found, currently active nodes, which is using that resource has its script invoked. It is then, matter of each script design to determine if what was performed

changed resource in any way important for given situation and whether to take any action, in a sense of changing resource state. Note that, from evolution system point of view it is completely plausible for one game entity to map to several narrative resources, which can introduce clash of interests for player. After script execution, integrity of current solution (see next section) is checked and control returned to game engine.

Next step after evolutionary consequences of player action, is to count consequences for player himself, expressed by fact that each activity character does in game costs energy, so energy cost is evaluated and player energy bar decreased. It is worth noting that whether player is able to pay needed energy cost is checked before action is executed and if that check fails, action results in failure, and player takes some minor damage. This mechanism is important because, for player, only possible way to restore energy is by sleeping. And sleeping takes quite a lot of time, which neatly moves us to next part of game event-loop.

Every action player is able to perform takes some time. It might be only a little, but is is still measured. Some actions, like sleeping take a lot of time, and this game time is what my system uses to direct evolution. This creates at least some sensation of order in game world with absence of any higher governing mechanism. Apart from that, it has quite nice effect of chance that sleeping player is quite inclined to be woken up by some event, generated by EA in vicinity. After elapsing of time period, game processing of player command ends, and system awaits for next command. Overview of game event-loop is presented on diagram 5.2.

Many aspects of game simulation are not implemented, or are only outlined in a way that allows working with EA and showing its properties. This includes for example player ability system, which is non-existent and replaced by set of commands. Another thing is combat processor, which is normally heavily relying on ability system. In this case it is substitutes by simple mathematical equation which takes properties of player, properties of hostile NPC, evaluates it and determines, with some stochasticity involved, outcome of a fight.

5.4 Evolutionary Algorithm

As I have mentioned in chapter 3, used solution is closest to Evolutionary programming in that it utilises completely domain-specific problem representation, and mutation operators for generating new population.

General outline of individual encoding was given in 4.3.1, so now we can concentrate on specifics. Object representing individual holds a vector of nodes and vector of resources, in addition to that object has a member representing goal identification, that is for which of available goals is algorithm trying to find compatible resource vector. Apart from these object has no notable members. We can move one step down and look how node and resource are represented.

Node hold pointers to resources it is using, this is important because node does not own the resources, it is just temporarily using them to change their state and help the narrative to move. Key property of node is status. This is enum which help, while operating over individual to easily tell which nodes interest us. This holds information whether node can be readily deployed in a game world and start interacting with player, or whether it is not ready yet, because it is waiting while player completes preceding node and frees the resources. Next, when node is finished, both in a sense player succesfully reaching on of output resource state combinations, and node being forcibly removed by evolution, its status marks it as such, so we can out-optimize such nodes from algorithm. It is however kept in

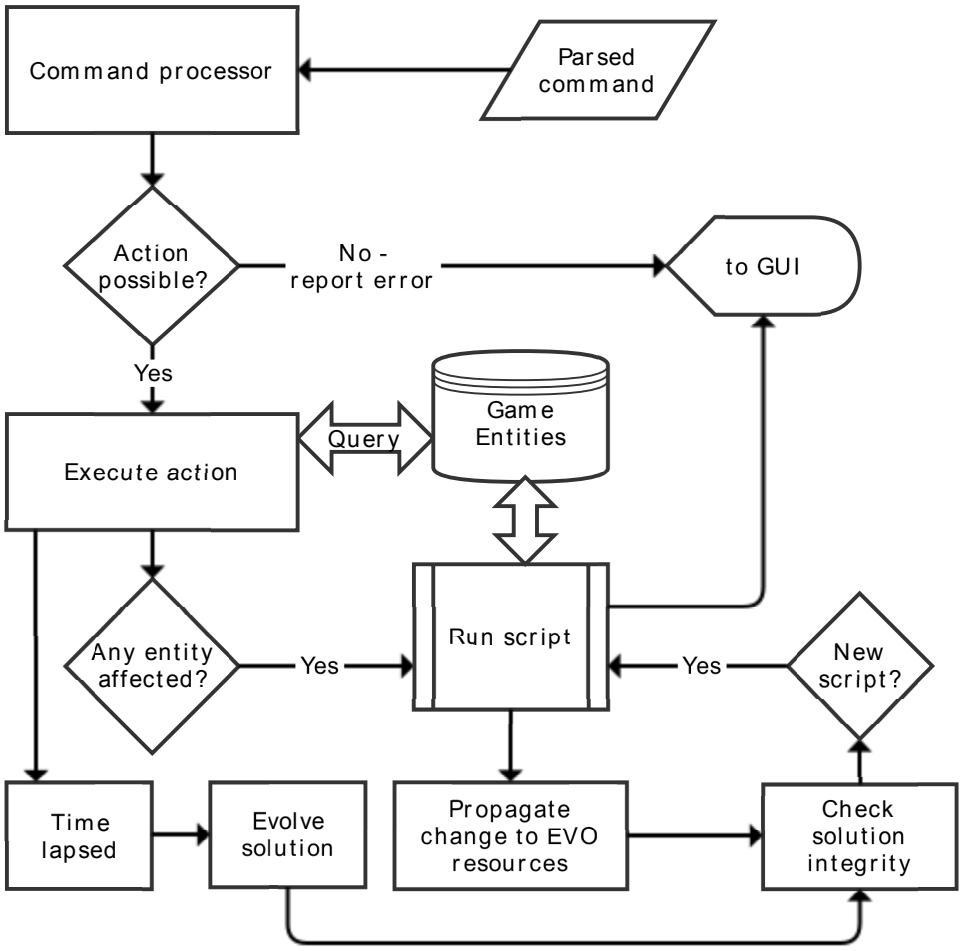


Figure 5.2: Diagram showing main game event-loop

nodes container, for logging purposes when narrative visualisation will be implemented and for possible future optimisations. These can include research about adapting fitness and operators throughout run, and for such things, information about history of calculation is often vital. Another property worth mentioning is age. As pointed out earlier, my solution uses game time to direct evolution, age counter of node starts with its construction and follows with game time increase, so this has double usage, once, to help determine node with "bad" input configuration which are in individual for very long time without ever reaching deployable state; second usage is for script associated with node, which allows it, upon being invoked by some trigger, to have a sense of time elapsed, thus giving possibility of appropriate reaction.

Next, we have a resource, which is object representing some game entity. As noted earlier one game entity can be represented by multiple resources, but each resources limited to being use in only one active node at any given time. This ensures nodes independence and prevents situation where whole solution evolvement could become easily blocked because of

one unluckily placed resource. Its main property is the status, which in abstract means expresses relationship of linked game entity with player. Beside it it has one more very important property and that is type. In implemented solution, currently only four types are used, NPC, Game-object, item and location. But designed narrative system works with these on abstract level, so is flexible to changes in this regard. Types are important because each script can refine what resources it requires, thus allowing to build more believable story. Resource types are also used while mutating in type-checking system, which ensures that only valid individual will make it to next population. Last thing which resource object holds is history of status changes. Currently only its length is used, but that may change in future optimisations.

Now, when we have specified individual representation, we may move to operators used. They are all three minor mutation operators, that is they have only one individual on input an they change one elementary part of said individual. In our case, first operator serves as creative, it has biggest probability of being used and function in a way that randomly adds one node to solution, while respecting all laws and limitations posed by individual. Second operator is reducing one, with lesser probability of being chosen. It uses optimised variant of roulette-wheel selection algorithm[9], which means that the higher is age of a node the higher chance of being removed node has. This helps to purge solution of nodes which have been generated with hardly satisfiable input configurations and are just computational burden. Last operator is special, have small chance of being used and only thing it does is, that it tries to satisfy conditions of placing final node in individual. This is conditioned by player character progress as well, so when selected in earlier stages of game this operator has no function.

The top-level function of evolution is relatively simple, it takes one argument which is available time, that is time player action takes. Each time "tick" equals to one population step. Step consist of creating `population_size` copies of current solution, and applying one - three random mutation to each generated individual. After mutating, integrity of each individual is checked and after that fitness evaluated. Selection uses same roulette-wheel approach as reducing operator, this time, of course by fitness, not age. After new individual is selected it becomes current solution and age of all nodes in selected individual is increased by one. This step is repeated until allocated time is depleted.

5.4.1 Fitness Function

Last topic left to introduce, concerning EA is how we determine which configuration of nodes leads to better story. Some concepts were outlined in 4.3.2. Compared to operators or ensuring individual integrity according to script constrains is relatively simple. First, we give advantage to solutions featuring more ready than not-ready nodes, because more ready nodes means more possibilities for player, which in turn results in him more precisely selecting what he would like, which help to next generations and so on. Finally this result in player having more fun which is certainly good quality of the story. Next, we want to have covered all possible outcomes, but to lesser extent than preceding criterion. So we give minor advantage to solutions which have higher coverage of possible node outcomes with incomes of another nodes. Another thing is that we want to best utilise available script library, so we compute a ration of existing nodes and available script and penalise solutions using one type of script (relative to the ratio) too often. Next thing comes that we count resources, but not by container, because those are same in every solution, but by references in nodes, and we give advantage to those with higher reference count of not zero

state resources (these will have to be yet generated), also this part is weighted according to whether node is ready or not and if it is a final node then points for these types of resources are multiplied even further.

5.5 Narrative Scripts

Last thing left to discuss are those scripts which will effectively be the only thing player actually sees. I will not go into great detail here because this work is focused on interactive narrative system rather then authoring stories, and only few most basic ones were implemented as proof-of-concept.

Script consists of two main parts, first is the actual body of the script to be invoked. This consist of game engine-level commands which influence game world. Those, while featuring small and simple point enough to be used as any part of larger whole, should be sufficiently interesting to make player act on it. Of course most scripts will feature some sort of reward as motivation for player, but it should not be relied solely upon that. Code of script has to have ability to determine context in which it is invoked and do appropriate action. Upon invocation, the system will tell script if this is the first time. If yes, than the script places hook for a player and add at least one trigger location. Next time it will be invoked either based on player entering one of marked locations or anyhow interacting with any game entity bound to resources used by said script node. State of those resources, respectively game entities, and time together should be enough information to determine context for trivial plot. Rest is completely open to implementation.

Second part of script is sort of header, placed as static variable, in which, along with script id, viable resources are listed. Form consist of entity/resource types tuple. Order matters here. This is one limitation of script, it has to operate with set number of resources of set types, and these cannot change throughout script. It is necessary for ensuring at least something around what I can design EA. Next there are available states of input resources, there is no limit on how much of them can be, but they cannot be combined, that is, states for all resources in one combination have to match for script to be placed and invoked. Therefore it is recommended that scripts work with smaller number of resources. Header continues with expected resource configuration upon script ending. As mentioned earlier script is not expected to anticipate every possible player action. It should just list few of the most sensible possibilities how it could end, so evolutionary story-planner has something to work with. If player acts out of plan, well, story will regenerate itself.

Ending scripts are almost like ordinary ones, just longer, it definitely should not consist of one if-then-else statement, player should be able to clearly notice it among normal happenings in game. As for being selected only once, it does not hurt if it require harder-to-satisfy combination of input resources. There is only imposed limitation, final script has one, and only one input configuration. This is for purpose of not allowing EA place it right away but creating need to prepare for it (final node has big fitness advantage). And understandingly, as it concludes the game, final script has no output configurations.

Chapter 6

Comparison with other approaches

In comparison we are going to use more general properties. Like some items listed in chapter 2, this solution could not be tested in real conditions (handed to users for collecting feedback). Because of lack of actual scripts which could be handed to EA to create real story, testing was limited to more abstract methods.

Most distinct difference between more traditional approaches and using EA is that there is truly no central structure, so before evolution pressure to repetead resource usage will start to be more apparent thing look very unorganised, and sometimes not make much sense. In the directed approaches the agent or used priciple is usually able to ensure relatively smooth progress right from start. Other problem is need to hold back the node generation so the player is not engulfed in consistent stream of new hooks being thrown at him. Another problem is convergence to some pivotal point in main story, which would conclude the game. This might be improved by fine-tuning evolution parameters when enough script for real story generation are possible.

On the other hand, Evolutionary approach offers immense replayability, when higher amount of scripts is available for EA to choose from, chances of even similar stories appearing anywhere near each other are very slim. Another plus point in using this evolutionary solution is its independence on domain. It just needs set of scripts and type classification of game entities. There is no direct interaction between game engine and evolution, except for concept of time, which is also only requirement for game engine design, because elapsed time is used for evolution steps. Concept of time helps yet in other way. As most of nodes feature some task for the player, with increasing time that player does not complete the task, probability increases that this task will be removed.

Chapter 7

Conclusion

In this thesis, we contemplated upon topic of the interactive narrative and several new interesting approaches to it. Rapid expansion of new fields in artificial intelligence has recently opened possibilities for developing storytelling system of significantly higher quality. Several currently existing systems, in different degree of completeness were listed, and their properties discussed. In detail it has been elaborated upon design of the interactive narrative based on evolutionary computing, more precisely evolutionary programming.

I designed a solution for storytelling, which utilises evolutionary approach to build a story. It uses set of scripts, created using game engine functions. These scripts provide abstract operation on game resources, thus reflecting change of relationship between player and game entity corresponding to said resource. These operations are performed based on player action, which is otherwise completely independent and not constrained in any way. Changes in resource state drive evolution and create selection pressure to build coherent script chains, which has emergent effect of game entities having their own defined behaviour, sometimes even one not ignorant of player, although this is nowhere explicitly designed. Another thing which is employed to, at least to some extent, make up for absence of any central planning is using time elapsed in game world to constrain evolution. As general consequence of more generations passed is more content generated, this helps to keep illusion of consistent world, which is in reality created by isolated nodes which have no knowledge about each other.

For presented design to be usable, a set of scripts with appropriate properties is needed. These elementary bits of story has to be general enough not to produce crooked construction when connected to other ones. There is room for improvement, for evolution engine to provide scripts with more backgroud information, which would result in more apt response regarding momentary situation.

Regarding possible expansions, these could include combination with different artificial intelligence techniques. This would help particularly at start, because at that time presented design behaves close to complete randomness. Method selected for such fusion should not be story director or other type of central planner, because such approach would probably interfere with evolutionary approach not leading to any recognisable improvement. Rather, something giving game entities, or parts and layers of game world itself some meaning and properties. For example, concept of relationships between entities, could be used, and expanding evolution operators with ability to utilize such relationships.

Further, provided we have enough scripts to use this solution at a larger-scale world, economic, social or similar model, or models, could be used to help govern actions of game entities when they are not defined by any currently active script. Such models could include

concept of balance and basically can be of two kinds. Either the player disturbs this balance and the system reacts in way to keep it, or, on the contrary, perhapes on larger scale, something else disturbs this balance and player must unite with the system or perish with it.

Another possible expansion is not so much computational, but more psychological. Different possible script outputs would, in addition to changing state of narrative resources, had associated certain *alignment*. This is concept used currently in majority tabletop role-playing games. It can be of several kinds, good–evil, lawful–chaotic etc. Alignment records would be kept for player character, based on which scripts ending would happen (caused by player actions). In tabletop games, this is usually used to reward player for acting correspondingly with his alignment chosen at start. But in our case, having node input and output configurations labeled with alignment distribution would be substantially improving factor to fitness evaluation. In addition to that, game entities would be able to react to player correspondingly to his prevailing alignment.

Concluding research done, solution designed and results currently available, an evolutionary approach can safely be deemed viable and appropriate for interactive narrative. While lacking in some areas, where combination with different approach or additional research and development would be needed, it can provide unique solution with very high replayability and good portability between story settings and even game engines.

Bibliography

[1] Alexandros Agapitos, Julian Togelius, Simon M. Lucas, Jürgen Schmidhuber, and Andreas Konstantinidis. Generating diverse opponents with multiobjective evolution. In *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games.*, 2008.

[2] Heather Barber. *Generator of Adaptive Dilemma-based Interactive Narratives.* PhD thesis, The University of York, 2008.

[3] David Beasley and J. Heitkötter. What's evolutionary programming (ep)? [online], Issue 9.1, released 12 April 2001, last visited 8.5.2015. available at: http://www.aip.de/~ast/EvolCompFAQ/Q1_2.htm.

[4] Karl Bergström. Framing storytelling with games. In *Interactive Storytelling:4th International Conference on Interactive Digital Storytelling*, pages 170–181. Springer, 2011. ISBN 978-3-642-25288-4.

[5] Kuthulan Erol. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation.* PhD thesis, University of Maryland, 1995.

[6] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Artificial Intelligence 2*, pages 189–208. North-Holland Publishing Company, 1971.

[7] Martin Gardner. Mathematical games – the fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

[8] Antonios Liapis, Hector P. Martinez, Julian Togelius, and Georgios N. Yannakakis. Adaptive game level creation through rank-based interactive evolution. In *IEEE Conference on Computational Intelligence in Games (CIG)*, pages 1–8. IEEE, 2013. ISBN 978-1-4673-5308-3.

[9] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica*, 399(6):2193–2196, 2012. doi:10.1016/j.physa.2011.12.004.

[10] Wizards of the Coast LLC. D&d official homepage. [online], last visited 5.5.2015. available at: http://dnd.wizards.com/.

[11] Mark O. Riedl and R. Michael Young. Narrative planning: Balancing plot and character. *Journal of Artifcial Intelligence Research*, 39:217–268, 2010.

[12] Lukáš Sekanina. *Evoluční hardware: od automatického generování patentovatelných invencí k sebemodifikujícím se strojům.* Praha: Academia, 2009. ISBN 978-80-200-1729-1.

[13] Gillian Smith, Ryan Anderson, Brian Kopleck, Zach Lindblad, Lauren Scott, Adam Wardell, Jim Whitehead, and Michael Mateas. Situating quests: Design patterns for quest and level design in role-playing games. In *Interactive Storytelling:4th International Conference on Interactive Digital Storytelling*, pages 170–181. Springer, 2011. ISBN 978-3-642-25288-4.

[14] Damian Sommer and Emily Carroll. The yawhg. [online], last visited 6.5.2015. available at: http://www.theyawhg.com/.

[15] R. Michael Young, Mark O. Riedl, Mark Branly, Arnav Jhala, R. J. Martin, and C. J. Saretto. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1:57–70, 2004.