

# Kingdom: Creating Dynamism through Quest Generation and Adaptive AI

## Participants:

Sriraman Subbaraman: [sriraman@seas.upenn.edu](mailto:sriraman@seas.upenn.edu)

Adam Porroni: [aap@seas.upenn.edu](mailto:aap@seas.upenn.edu)

Spencer Miller: [spencerm@sas.upenn.edu](mailto:spencerm@sas.upenn.edu)

## Faculty Advisor:

Professor Norman Badler

## Background and Abstract:

Computer games hold a special place as a key area that spurs the development of better artificial intelligence and algorithms and structures that all go towards providing a more engaging and challenging experience for the player.

In many games, the aspect of simulation with respect to active AI is overlooked, due to a lack of emphasis from game studios, with only a few notable exceptions. F.E.A.R., one such exception, implements adaptive AI in the player's opponents, who react to strategies the player employs and then counter with strategies that are effective in defeating the player. In addition, this system allows difficulty to be solely based on the player's first-person shooter (in this case) playing ability, as detection of novice playing can be also be adapted to by toning down the level of the AI. This makes for a much more accessible game that also feels much more realistic, as enemies dive behind cover and initiates circling maneuvers in order to kill the player. The other game that comes to mind is The Sims, a game in which the lives of the Sims are computer generated but player guided. Each Sim character has a set of needs and a life of its own, and the player must deal with those needs. This creates a dynamic world that the player must react to, allowing a player to get much more involved, than in other games. However, neither of these games taps the true potential of AI controlled NPCs or objects, or of adaptive AI. This is the basis for our research.

In addition, games revolve around a set of central concepts, one of which is suspension of disbelief, the barrier between a player being truly involved in the game and distancing themselves from it because they cannot suspend their disbelief. Suspension of disbelief requires that a game follow common sense, the common sense of real life. In the field of role-playing games, especially Massively Multiplayer Online Role-Playing Games, the presence of **static** non player characters, or NPCs, will erode away any initial suspension of disbelief. This is the key factor in a problem called "grinding", which is essentially the fact that most RPG's have you do the same quests for different static NPCs in order to get experience. Only a small subset of game players is willing to stand the grind, and no solution has surfaced.

The goal of this project is to add dynamic qualities to a gameworld in order to establish a greater level of depth and fundamentally redefine how gameplay works. We will design and implement “Dynamic” NPCs (non-player characters). A Dynamic NPC or DNPC, will operate as a thinking entity in the gameworld. Each DNPC will have an AI that attempts to accomplish the tasks generated by the role of the character. In attempting to fulfill a role, the DNPC may need to accomplish a task well outside its skill set, and at that point it would spawn a player quest and the content related to that quest. This system creates a dynamic world for the player to play in, one that changes with and more importantly without the player's direct input. In addition, because quests become things necessary for the success of the DNPCs, they make much more sense to the player, and can provide a greater sense of accomplishment. However, the greatest consequence of such a system is the evolution of game world simulation, from a fundamentally static environment to a dynamic one. Indeed such a product would not only bring games to the next level, it would provide an interesting platform in which any number of social simulations could be run using those DNPCs. In fact, the AI controlling those DNPCs can undergo almost infinite revisions as the level of machine learning algorithms and adaptive AI techniques increases, as those two fields combine to form the character of a DNPC.

In sum, our task is to make a “Kingdom” of DNPCs, a self-contained and yet self-sufficient community of dynamic characters within which a human player can immerse himself or herself. This community will not require the player as implied above, but rather will be augmented, complemented by a single or many human players interacting among the DNPCs.

### **Related Work:**

Toward the goal of developing a fully immersive environment could be argued as the eternal trajectory of game evolution and design. With the fantastic advances in CPU processing potential, memory storage schemes and even GPU – graphical processing unit – technology, computer, console and arcade games have never before been as exciting and realistic as they are today. Alongside these developments comes desire, for, as Dr. Alexander Nareyek notes in the January 2007 edition of IEEE Intelligent Systems AI magazine, game-players fully appreciate these advances and “want ever-increasing free-roaming environments” at their disposal (Nareyek, A., 2007). These demands have not fallen on deaf ears, for industry Goliaths and Davids alike are searching for better and better storytellers to write their plotlines and adept artists to design their landscapes. However, except for some well-respected outliers in the game development industry and academia, research and production of realistic artificial intelligence systems to augment potentially immersive games barely reaches the mark.

Among the exceptional minds researching interactive, proactive game AI is Dr. Pieter Spronck, a Dutch computer scientist who has produced dozens of papers and experiments on AI best practices in game environments. Currently working at the University of Maastricht in their esteemed Institute for Knowledge and Agent Technology, Dr. Spronck has set his focus mostly on dynamic adversarial AI models for real-time strategy and, just recently, role-playing games (see Spronck, P., *Phase-*

*Dependent Evaluation in RTS Games*, (BNAIC 2007) 3-10). Among his most exhaustive research experiments has been in dynamic scripting, where he has explored diverse schemes for autonomous decision making by the game AI. His major paper in this research is *Automatic Rule Ordering for Dynamic Scripting*, which tests the effectiveness of a weight-based priority system for ordering character combat commands in a RPG battle (Spronck, P. et al., 2007). Most pertinent to our own work, he developed this dynamic combat AI using the *Neverwinter Nights* toolset. However, it is purely an adversarial and goal-oriented AI, which differs from our proposal in that his NPCs have a far more limited decision tree. Likewise, his models provide no way to interact with the characters in a peaceful way and do not provide the extensive quest creation algorithms that we will produce in full.

Other leaders in the field of game AI focus largely on perception and knowledge acquisition in order to tackle the machine learning challenge. Thore Graepel, Ralf Herbrich and Julian Gold have all produced a machine learning system specific to winning one on one game combat with a pre-made game AI, which they catalog in an important work, *Learning to Fight* (Graepel et al., 2004). This system of course perceived the enemy character, but the central experiment in this research concerned knowledge acquisition and reaction to rewards when the AI scored hits against its opponent. In this case their algorithm followed the Markov principle where the amount of data required to store the game AI's state never changed and the AI scheme could effectively remember and learn from past behavior. Their success, though they admit it was limited, provides us with important insight into the use of finite state machine (FSM) schemes that can potentially implement complex quasi-thinking in an assumed deterministic learning environment.

Another of Dr. Spronck's research papers has also become an important influence on our work and design principles. He, along with Sander Bakkes, explored in 2005 the possibility of approximating a group of separate AI decision making processes (i.e. for separate characters in a game realm) with one AI for the team as a whole (Bakkes, S., Spronck, P., *Symbiotic Learning in Commercial Computer Games* (CGAMES 2005), 116-120). This paper also did not focus on role-playing games, but "parties" or teams of characters joined together to accomplish a task, are common in RPGs. The significance of this research is its emergent behavior, which we are also trying to capture as the "Kingdom" develops, since the team controlled by one super-AI developed successful strategies that took advantage of its unit's cohesion and regularly overwhelmed its opponent teams. Perhaps the best result of this research for our work is that we can, when our DNPCs join into a party, suspend their respective AI from excessively dynamic behavior in order to accomplish the task at hand more effectively than when they would solve the problem separately.

Interestingly enough, our research has not been solely in the realm of computer science. In fact, we did not have to look far in order to learn about emergent behavior and genetic, or evolutionary, algorithms since Ian Lustick in the University of Pennsylvania Department of Political Science has worked for years on exploring this topic. His platform, PS-I or ABIR, for "agent based identity repertoire" model, combines more straightforward principles of AI development with agile evolving environments. One of his signature papers, published along with A. Maurits van der Veen and Dan Miodownik and titled *Studying Performance and Learning with ABIR: The Effects of*

*Knowledge, Mobilizing Agents, and Predictability*, surveyed their work in simulating political environments using rather basic agents with simple AI evolution rules (van der Veen et al., 2001). The central idea that we have taken from this work is that for large environments with similarly large communities, an extremely simple rule base can produce results that mirror nearly human political interaction.

Our proposal is audacious, but could be highly influential in both the game AI field and the greater field of dynamic machine learning and emergent systems. At the very least, our system will provide the user or player with an immersive environment where the dynamic non-player characters interact with other characters at their own pace by their own decisions. All too often, as mentioned earlier, players in games such as *The Elder Scrolls* or *Neverwinter Nights* wander around a world without any proactive adversary. The game experience is mostly wasted time as the player has to seek out quest-giving NPCs in order to further a desired storyline. The norm in this case is not innovative or much of an experience at all, and our system seeks to change that paradigm.

This kind of system will provide a complete suspension of disbelief as a player can walk around this “Kingdom” as he or she would on a normal Philadelphia street since life in this realm can move along without his or her direct influence. The landscape would be thus dynamic to a degree never before experienced in virtual gameplay. In this sense *alone* we will provide a completely unique game environment, but our model goes further in that the DNPCs can interact with each other, work together or against one another, and choose to follow paths similar to what has always been allowed only to the player. In this way the user or player can simply observe their behavior, and thus the older paradigm shifts, for it is no longer the player who is the center of the universe, but choice itself.

### Technical Approach:

The system contains two main elements that are significantly interconnected. Those elements are quest generation and dynamic NPCs. Implementing one requires the use of the other in order for any true system to be established. Our main layout, provided below, has the caveat that each of those boxes obviously contains further layers of complexity.

- AI controller: Deals with the interaction between other aspects of the game, including environment change, and communicate them to all of the DNPCs. DNPCs will also send quest generation requests to the controller, which will send those requests to the quest generator and then handle the response, sending the pertinent information to the original DNPC.
- DNPC AI: These AIs will have to determine the action of a DNPC at the time that either a game error is encountered (a blacksmith runs out of material in middle of smithing), or when an AI finishes their current task. Most likely, a module would be a task manager or some such that allows the AI to keep track of tasks it must undertake. This AI would have either to be a complicated state machine implementation or have a dynamic scripting tool generate the AI’s action list.
- Quest Generator: Uses a formula in combination with the domain of the AI that is initiating the generation to generate a quest. In higher levels of implementation

the quest generator will use an AI's 'network' (List of DNPCs the AI's DNPC knows) to check if another DNPC can complete the quest, if so, then it passes that quest to the pertinent AI through the controller.

- Instantiator: At base level of implementation, it generates the material required to complete a quest. It generates no in game content past the required material placed in a default, pre-existing location. Higher implementations would generate instances for quests that take primarily the player outside of the region that the DNPC resides. E.g., Player must go deep into forest to retrieve some rare herbs, forest/forest entrance is pre-existing, the deep woods instance and entrance are generated and then destroyed upon full quest completion.
- Domain: This element exists under each DNPC, and contains both the state and the defining characteristics of the DNPC. This domain provides the set of rules by which the AI is governed. The AI controller would have to update the domain as the environment changed, and quest completions would change the domain and thus result in progress for the DNPC.

Heartsong – Heartsong is the engine that determines the next choice of action during a heartbeat in which the DNPC chooses to do something else. Our main issue here was actually developing a method by which decisions would not be deterministic and thus convergent when thinking about behavior. All of the related work in this area is about AIs in more strategic game like environments, like chess, in which there are clear notions of doing better and worse, and success and failure, versus the lives of our DNPCs for which judging optimality becomes an issue. Heartsong will most likely take a weight-based look and either generates and/or acts upon desires, those desires representing the rules by which our state machine will operate. At least evolutionary learning will be represented as we can modify weights of certain actions/desires as per the consequences of acting on those desires. This issue has not been solved yet, and will most likely be more dealt with once we implement everything and see what actually emerges out of different rule-schemes. This however is not problematic, as we have used a modular design that will allow significant modification to the Heartsong engine without too much grief. This specifically comes into play with the Verb engine, which will compile the Heartsong result into a script that NVN2 will actually be able to understand, since having such things specific to an implementation of Heartsong would mean headaches if we had to reinvent Heartsong later.

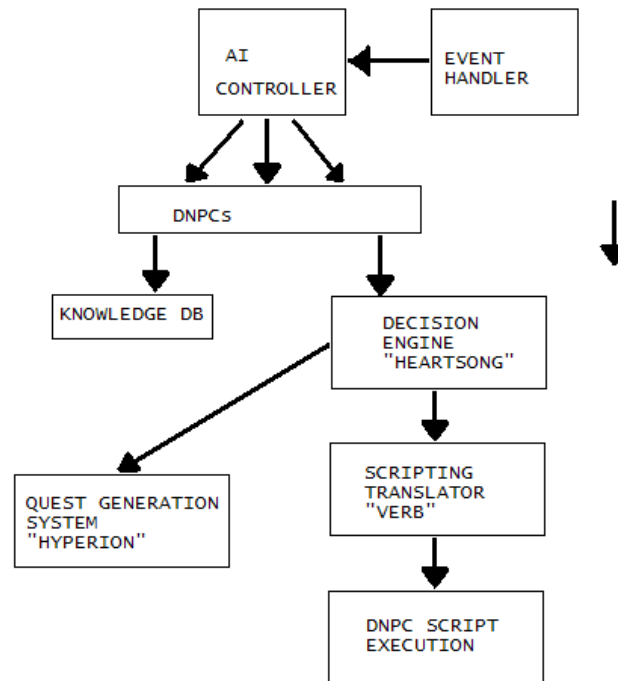
Domain – We rethought domain in terms of what variables define a character or a person, and further more the difference between variables internal and external to the character. In order to have an attempt at the discussed growth of a society there were some concepts we had to come up with, like the general societal knowledge, which especially comes into play in terms of the evolution of societies. Domain information will work as the data sent in between the various modules of our dynamic scripting machine, with different modules either updating or utilizing state information in order to progress the DNPC.

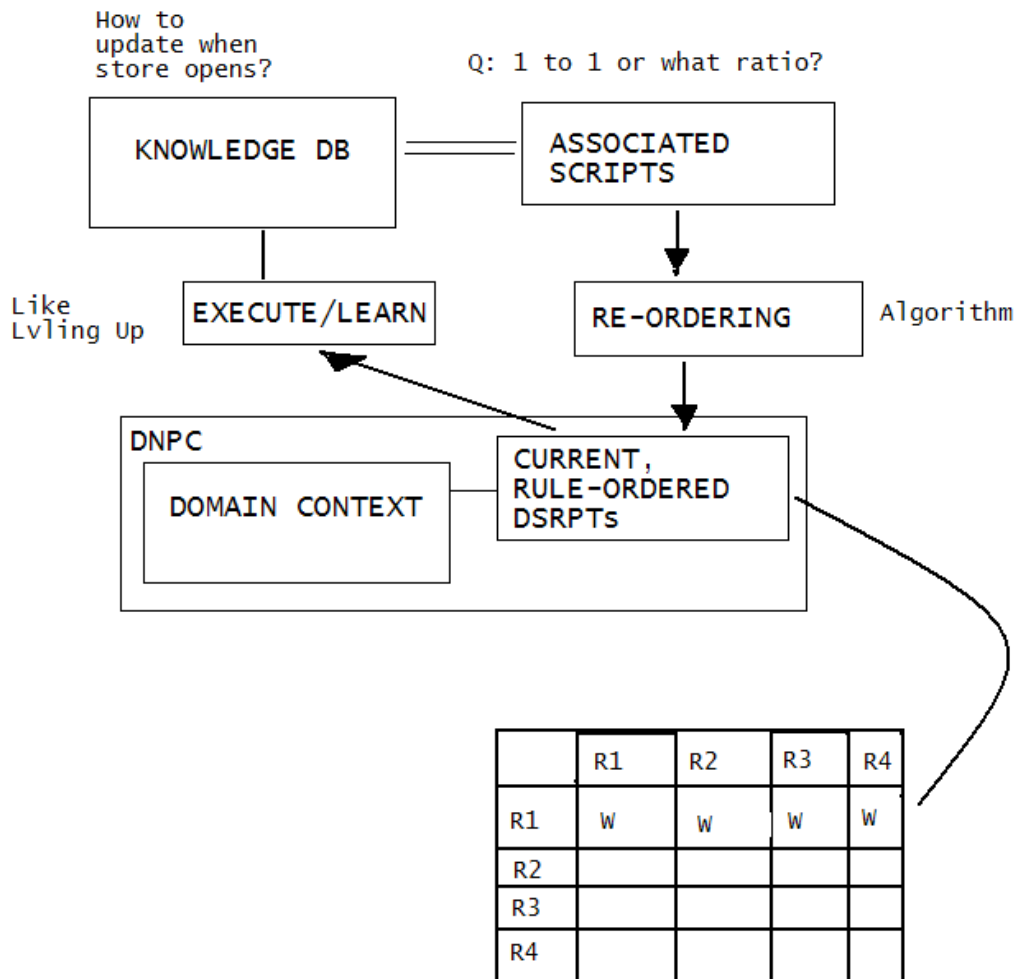
AI Controller – Simply an interface between the DNPCs, the environment and the quest generator, it will allow changes in the environment to be quickly passed down to all pertinent AIs, and will also handle communication between AIs. This will be done

using relatively simple comparisons of state in order to see if the interaction between DNPCs is positive, negative, or non-conclusive.

In general, much of our technical work has been really laying out and designing the project so that we are not just sitting down and coding. Much of the design issues we have encountered come from attempting to work in implementation of the ideas discussed during our first review, like knowledge generation and emergent behavior. Especially emergence requires that we not willingly determine behavior by a set of predefined rules, but rather that through machine learning and other probabilistic determinations, DNPCs will build their own preferences and actions.

Additionally, we have had to master the Aurora toolset and NWScript, both of which have a steep learning curve for those not accustomed to modification environments. One such necessary step was the step by step progress through the NWN2 Builders Guide, a 233 page work on all of the elements of the toolset including scripting and how best to use those scripts. Also, there are many elements of the NWN2 engine and interface that we are attempting to master in order to streamline production of our project.





Based on Model on RWO+B in Spronck et al. AIIDE 2007.

### Resources Required:

- Our respective computers
- The Aurora Toolset (Neverwinter Nights 2 Toolset)
- Aurora Toolset Guides/Tutorials

Possibly:

- Books on AI design, scripting
- Books on Game Development
- Books on C# (Aurora is written in C#)

## 1 WORK PLAN

---

### 1.1 Tasks

### **1.1.1 Alpha Version**

Our goal for the alpha version is to have a simple working model of the Heartsong and Verb engines for an arbitrary amount of acting DNPCs. Available actions will be rudimentary, with room for expansion to the two engines. Our Hyperion (environmental control) engine will be left out of alpha, as it is not a necessary component to independent actors.

### **1.1.2 Beta Version**

Our goal for beta is to fully flesh out the actions available through Heartsong and Verb and to implement the basic features of the Hyperion engine (that being, control of the “common knowledge” pool). With the release of beta, DNPCs should be much more lifelike than those of alpha, performing realistic actions you would expect of a thinking actor.

### **1.1.3 Final Version**

The final version will be an optimization of the beta, also including any additional features that we have time for, such as: item generation, knowledge generation, and environment generation. It will have a full featured domain.

## **1.2 Schedule (by team-member if applicable)**

Week of Feb. 3 – Master toolset, begin scripting

Week of Feb. 10 – Begin Alpha work

Week of Feb. 17 – Complete Alpha, begin transition to Beta

Week of Feb. 24 – Work on Beta

Week of March 2 – Complete Beta, begin transition to Final

Week of March 9 – [[Spring break]]

Week of March 16 – Work on code optimization

Week of March 23 – Attempt additional features (Knowledge, Instance Generation)

From end of March to end of April – continue work on optimization, additional features

## **Conclusion:**

We all knew that this project was audacious, but when we did our initial research and, while brainstorming, we fully appreciated its complexity. AI schemes involve degrees/challenges of machine learning, reinforcement learning, (sometimes these terms are equivalent, but the latter has a social-psychological background to it while the former is defined in discrete mathematical and computer science terms), efficient memory storage, dynamic scripting, tactics, autonomous planning, and perception. In game environments, AI schemes have to incorporate most if not all of these parts in a way that makes the game interactive, believable, and fun. Therefore, there is a major technical aspect to AI designs, but there is also a significant creative aspect.

We had to research and brainstorm about a broad range of topics. We divided the work into a few parts - such as schemes of emergent behavior in deterministic systems, machine perception in changing environments, and dynamic scripting in turn-based game engines. We found papers, read them, and thought about our own way of combining the

different aspects. Once we thought of arguably testable schemes in the Neverwinter Nights toolset, we began contrasting and combining our ideas.

We followed the same research-then-discuss-then-redesign plan for the rest of the project, where most of the focus was on autonomous decision and planning. Since this subject is one in which there are so few established/accepted best practices, we began to do thought experiments using our own experiences in "what makes us want/desire to do something". To that end, we tried to break down how something decides.

In our initial attempts to define our project, we decided that design would play a significant role in being able to efficiently and effectively code our artificial intelligence. We determined that our DNPC artificial intelligence was essentially modeling human behavior, so we initially intended to fully lay out our design for the AI and then translate that design into clean and modular code. A good part of our generated work product lies in this robust analysis of modeling human behavior, in an attempt to add enough complexity to our AI in order to gain behavioral emergence and create an interesting virtual society. Our design began with laying out the factors and influences that define a human's existence, and then attempting to build those factors into our system of weighted rules that would allow us to dynamically script. This was the most time consuming part of the design process, as defining a character's domain was providing both the state that would exhibit the growth and learning a DNPC underwent, and provide the weights to differentiate character actions during the Heartsong process.

In our domain design, we were exhaustive in our definition because this would allow us create an iterative plan for coding the DNPC AI. We broke up the elements in a character into two fields, internal and external variables. Our beginning focus was on the internal variables of character statistics, needs/wants, desire, and the external variable of friends. These elements would provide a basic domain from which an AI could have multiple desires and multiple ways to achieve those goals. Every variable we defined made up a unique attribute that could be weighted and added to the calculus for decision-making. As we started to focus on creating a realistic initial prototype, we took those beginning elements and defined our domain in terms of what the actual values and weights would be. In specific, we focused on the basic statistics of a character, and sought to define them in a way that would be meaningful in our calculus. That meant taking a typical strength statistic and breaking it up into upper body and lower body strength, factors that could be used in the calculation of ability to do some task during a Heartsong call. Each following statistic was analyzed to see if it was necessary or if it could be replaced with a unique identifier. In addition to basic statistics, we dealt with the needs/wants entity in our domain.

We divided this into basic needs and wants or desires. We defined our characters basic needs along the lines of Maslow's hierarchy and our own impressions. Our basic needs were hunger, thirst, fatigue and, well, *pooping*. These form a very basic set of entities that require decision-making attention from a DNPC. Coding this would be our first step. After that, we attempted to address desires, and we found that it was impossible to list desires and weight them without arbitrarily defining a character in a deterministic fashion, which would hinder behavioral emergence. Thus, we conceived of 'drivers' or criteria that determine the desirability of something. We initially defined four such drivers. The first is joy, defined as the gratification, happiness and or satisfaction

something brings. The next is pain, the physical or mental stress caused by something. There is also money, which is our representation of worth and opportunity of something. Finally, there is the social driver, which represents the social impetus that humans have towards social interaction. These drivers would be weights on a range of activities and actions and a character's basic traits that deal with these drivers, such as pain tolerance, would allow for a diversity of behaviors.

Another phase of design that occurred closer to when we were converting our initial AI plan into C# code was defining and modularizing the Heartsong engine into two distinct components. We determined that every action is the resultant of two phases, the decision phase and the planning phase. The decision phase is a character asking itself, 'what do I want to do', and running through its current actions and then its desires to determine if a new action is desirable enough to put down the current one. Once the character chooses a desire, it enters the planning phase, in which a character now asks, 'how do I achieve this desire'. All of these choices are made through a system of weighted rule bases, in which the weights differ between characters, and change themselves. Planning consists of keeping a set of ordered 'rules', in this case, in game actions, and then having a weight on orderings of those rules that represents the success of that plan in achieving its goal. In this way, characters will steadily arrive at better plans in terms of achieving what they seek. Our initial implementation of this design dealt with simply basic needs, deciding which need should be addressed, if any, and then planning on how to get it, by considering the locations they know and the contacts they have in relation to the desire in mind. The highest weighted plan would be executed, and the resultant decrease in the associated need variable would be used to weight, positively, the used plan.

We planned to code this project in an object-oriented environment before we began our research. OO programming would allow us to modularize our code effectively in order to reduce the difficulty of debugging, divide the work and aid in the initial design process since we could visualize the parts working together. Each of these benefits was attractive since we all had little experience with AI but extensive experience in programming large projects. So we set out to find research in Java, C# or even C++, all of which are either explicitly an OO system or can simulate one through data structures.

However, we also thought we should be grounded in a game environment, and most of our research pointed us to Neverwinter Nights. As an RPG with a campaign editor, it gave us the option of a fancy 3D environment alongside what we assumed to be a robust scripting and programming environment. In fact, the papers we read explained some seemingly high-level interactions between NPCs, but that initial assessment soon proved false. Even though the toolset could nominally compile and interact with C# code, and thus we did think that we could port our C# code into the Neverwinter system and run it parallel to the toolset's native NPC processes, it could not maintain persistent blocks of memory for variables and had no capability of understanding arrays or other data structures. So, even though we built significant portions of the project entirely within the toolset's own scripting language, we decided to scrap it later due to those restrictions.

Now we are porting over our designs and algorithms the way we conceived them in a Java environment after we researched two other graphics engines. Java is completely object oriented and using a pre-designed graphics environment allows us to just port over our AI scheme without any pre-conceived game rules and restrictions. We will design

the DNPCs exactly as desired and we will create the visual experience, even if it is not as fancy as a commercial game.

The single greatest difficulty we faced is that the area of artificial intelligence is largely unexplored, and the scant papers available are only distantly relevant to the environment we were trying to create. In such a situation, the burden of research falls to the designer, who must muck his way through trial and error towards a palatable solution. Many design paths we chose turned out to be dead ends, and it was only after witnessing the results first hand were we able to discount them appropriately. A pertinent example would be our original system of handling character thoughts - nicknamed "MindBags" due to its reliance on a character's inventory. The MindBag system would track a character's current thoughts and desires as physical items in his inventory. He could have an "I WANT TO EAT" item, an "I WANT TO ALLEVIATE MY BOREDOM" item, and an "I WANT SOME FLOWERS" item, all given weights according to how important they are to the character. As the character satisfied these desires, the items would either have their weights reduced or be removed from the inventory entirely. The system also offered the bonus of an easy transfer of desires and tasks between characters. For example, if one character wanted to give the task of gathering flowers to another character, he could simply transfer the "I WANT SOME FLOWERS" item directly to the second character, who would in turn generate an "I WANT TO GIVE SOME FLOWERS BACK TO [CHARACTER A]" item.

The first roadblock we encountered with MindBags was a concern with their efficiency. It seemed like creating, holding, and transferring physical items in game was forcing unnecessary book keeping on the engine, and may demand too much time and memory from the game to be feasible. Later, after we had actually begun the implementation of MindBags, we discovered that the Neverwinter engine did not support certain calls necessary to the system, and so it fell apart entirely. In fact, we could not find any reasonable way to access an arbitrary slot in an inventory, so array simulation and variable-based loop iteration became unfeasible.

This grappling with the toolset was not confined to MindBags. In fact, the limitations of the toolset stand as the other set of Great Difficulties between grandeur and us. The Neverwinter toolset contains a proprietary scripting engine, which itself seems to be C++ based, with the following restrictions: no collections of any kind (arrays, lists, etc), no classes or support for object-oriented programming, no persistent global variables, and the list goes on. What this means is that all programming practices that we had studied in our courses up through this year no longer applied, and we were faced with the harrowing possibility of cramming big square pegs through small round holes if we wanted our AI system to work in the Neverwinter engine. Obviously, the choice of implementing engine was not the important one, and so we eventually decided to ditch Neverwinter in favor of our own graphics engine that could operate in a standard, supported, and tested language.

While we had anticipated difficulties with the toolset, and planned to counter them by writing the majority in our code in C# that would run parallel to the Neverwinter script. The C# code would control the data and process updates, and then generate Neverwinter script-readable code, which we would pass through to the Neverwinter engine. Unfortunately, things again ran afoul, and we discovered it was impossible to dynamically load a script into Neverwinter while it was running, as all scripts required

precompilation through the Neverwinter linker. Similarly, even if we provided the toolset with a precompiled script, we needed extra diligence in the bookkeeping of our variables to ensure that the memory block would not be erased due to an improper call.

Unfortunately, at this stage in development we do not have the intelligent interactivity that we wanted. Much of the project was spent exploring various avenues of implementation, and now the goal has shifted to include developing an environment that can support the design plans we had originally laid out. Our successes within Neverwinter extend to having a character that is able to track and respond to his basic needs, but no further. The decision to shift to a more open environment will allow us a vastly improved potential for interaction, limited only by the serial nature of programming itself.

The new destination platform will be a modified version of Spencer's scene graph, which was coded last semester in Computer Graphics. The scene graph acts as a 3D environment that is able to track any objects created within it. It supports four types of primitives through OpenGL (spheres, cuboids, planes and lines) and includes support for .obj files created in popular modeling software such as Autodesk's MAYA and 3DStudioMax. For the purposes of the Kingdom project, we can represent DNPCs within the engine as, for example, spheres, and then have the spheres move about the world according to the state of their desires and actions. We can easily track the entire internal state of each DNPC, something we could not do in Neverwinter, and have that translate to easily recognizable actions on screen (though, admittedly, these actions will be less visually complex than those included with the Neverwinter engine).

## Related Work:

*Neverwinter Nights 2 Vault*. (n.d.). Retrieved September 12, 2007 from <http://nwwvault.ign.com/>

Neverwinter Nights 2 Vault is one of the coding resources we are using to get general information about using NWScript and the NWN2 toolset, and additionally specific answers to certain issues that may arise as we code the project. The site provides key community resources in addition to open-source scripts that can provide some functionality that we could use to code faster, such as toolset plug-ins that help with world design and script attachment.

*Aurora Toolset*. (2007). Retrieved September 12, 2007 from [http://en.wikipedia.org/wiki/Aurora\\_toolset](http://en.wikipedia.org/wiki/Aurora_toolset)

The Aurora Toolset wiki provided important background information about the toolset that we are currently using in order to implement Kingdom. We had originally been confused about whether the Aurora toolset was actually the toolset for NWN2 or if it was only the toolset for the original NWN. The wiki provided a lot of useful background information that cleared up things like this.

Adamson House, Ltd. (2007). *Neverwinter Nights 2 Scripting Site*. Retrieved September

12, 2007 from <http://www.nwn2scripting.com>

The NwN2 Scripting Site provides an in-depth tutorial on scripting as it pertains to NWScript and actually generating scripts to use in modules. This is the brunt of the study we did during the first semester, as we needed to be very comfortable with both the concept of scripting and also writing NWScript before attempting to write the Verb engine that would generate NWScripts itself.

Altrico, Ltd. (2006). *Neverwinter Nights 2 Toolset Guide*. Retrieved September 12, 2007 from <http://www.nwn2toolset.com/index.html>

The Neverwinter Nights 2 toolset guide site provides resources for understanding how to use the Neverwinter 2 toolset and tips and trick that make product generation faster. It contains tutorials that cover basic toolset functionality and a full-fledged builder's guide that takes a more in-depth step by step look at the toolset features and how to use them. The builder's guide was especially useful in mastering the Aurora toolset and beginning to understand the elements required to seed worlds with characters and other aspects of content generation and its feasibility.

Bioware, (n.d.). *Neverwinter Nights 2 Toolset and Modification Forums*. Retrieved September 12, 2007.

<http://nwn2forums.bioware.com/forums/viewforum.html?forum=100>

The toolset and mod forums provide a much more targeted source of help, in that searching its many threads and replies allows us to find answers to more specific questions that the guide did not generally cover. In the forums there are many scripting issues and examples that not only gave us an idea of certain issues that could arise as we coded, but also gave us valuable information specifically about scripting in terms of AI.

Timor, T., Spronck, P., & van den Herik, J. (2007). Automatic Rule Ordering for Dynamic Scripting. *Proceedings, The Third Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE07)* (eds. Jonathan Schaeffer and Michael Mateas), 49-54. AAAI Press, Menlo Park, CA.

This conference paper, presented for the 2007 AIIDE conference sponsored by the scholarly Association for the Advancement of Artificial Intelligence (AAAI), is an academic review of experiments in script ordering schemes. Though scholars all, Spronck especially is one of the leading researchers in the field of game AI. This paper grants the reader some critical insight on potential schemes of rule-ordering, which is used to organize dynamic scripting events in order to mimic logical human decisions of in-game play. Of particular note is that the three schemes addressed in the paper automatically can, with a similar knowledge base available to the AI, compete with the efficiency of manually inputted ordering of character commands. The paper is also a truly scholarly work, for the research team acknowledges the pitfalls of relying too heavily on even their best schemes, as long-term play with a larger set of command options may lower the overall efficiencies of computer characters. They leave this caveat

since research in this field of adaptive, dynamic game AI is still nascent. Lastly, this work is extremely pertinent to our project since similar script-ordering techniques may give our dynamic non-player characters the ability to make much less chaotic or unreasonable actions in given situations.

Graepel, T., Herbrich, R., & Gold, J. (2004). Learning to Fight. *Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)* (eds. Mehdi, Q., Gough, N., Natkin, S., & Al-Dabass, D.), 193–200. Wolverhampton, UK: University of Wolverhampton.

A useful, but, the scholars admit, incomplete academic work presented during the 2004 CGAIDE conference of European computer scientists, this paper discusses the qualities of reinforcement learning on adversarial combat games. Specific to one combat game for the Xbox console, the researchers tested a modified SARSA algorithm (and “on-policy learning algorithm,” pg. 4) for their game AI to follow when combating the game’s predetermined AI rules in the fighting game. The experiments involved four different overall strategies for their modified AI to follow that were either rewarded or penalized. In three separate groups of experiments, the researchers effectively showed how varying benefits can create an environment for good learning policies. Some simpler emergence also arose, for the modified AI actually exploited, on average over many simulations, pre-set advantages provided to a would-be player in certain modes of the game. Though the research is still on-going as the scholars acknowledge, this earlier work helps us to understand the potential of simple schemes to produce near-human like behavior.

van der Veen, A. M., Lustick, I. S., & Miodownik, D. (2001). Studying Performance and Learning with ABIR: The Effects of Knowledge, Mobilizing Agents, and Predictability. *Social Science Computer Review* 19(3): 263-279.

An early, but original work by Ph.D.-awarded scholars van der Veen, Lustick and Miodownik, this research paper surveys and catalogs experiments in the Agent-Based Identity Repertoire (ABIR) modeling platform developed at the University of Pennsylvania. A part-theoretical, part-scientific model, ABIR allows users to implement diverse types of populations and how they can interact by virtue of its base unit, the agent, and its numerical set of identities. This paper explores the varying weight of influences on the landscapes of the ABIR model. For instance, much care and academic rigor is placed into proving how the integral aspects of the program and model do not predetermine agent activity, but rather enhance unpredictability toward dynamic equilibrium states. For the remainder of the research paper, increasingly complex landscapes of agents are compared in performance to simpler ones and the results are quite clear, namely that keeping environmental rules simple and clear maintains reasonable stability of landscapes while keeping the possibility for strongly emergent behavior. Our interest in this work stems mostly from its theoretical example, that

computer models and simple AI algorithms can still produce complex behavior and results.

Bakkes, S., & Spronck, P. (2005). Symbiotic Learning in Commercial Computer Games. *The 7th International Conference on Computer Games (CGAMES 2005)* (eds. Mehdi, Q., Gough, N., & Natkin, S.), 116-120. University of Wolverhampton, School of Computing and Information Technologies, UK.

This research paper had been presented during the International Conference on Computer Games in 2005, which is regularly sponsored by the IEEE Society and covers the many animation, AI and graphical processing topics concerning modern gaming. Spronck, among the most prolific academic authors on the subject of game-related AI schemes, joins with a doctoral student, Sander Bakkes, to produce this work on “adaptive team behaviour” (pg. 1 of the original work), that is an AI scheme that can treat a squad of characters as one, cohesive, goal-driven unit. They name this scheme “symbiotic learning,” which they develop as an adversarial AI governing opponents in a “capture the flag” game in the QUAKE III environment. The bulk of the experimental work featured in the paper involves simulation running where one team of QUAKE bots is pitted against a team of symbiotic non-playable bots. The results demonstrate that the resource efficient team AI method successfully approximates a team of disparate bots controlled by separate AI. In fact, and most pertinent to our work, these experiments actually presented an aspect of emergent behavior, for the symbiotic bots were more likely to engage in more offensive tactics when it was apparent that they team was more organized. Due to our own desire to produce a realm disposed toward emergent behavior, this relatively simple example is encouraging.

Bakkes, S., Spronck, P., & van den Herik, J. (2007). Phase-dependent Evaluation in RTS games. *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)* (eds. Dastani, M. M., & de Jong, E.), 3-10. Universiteit Utrecht, The Netherlands.

Presented during a 2007 regional scholarly conference, where esteemed Belgian and Dutch research universities are represented, this paper explores the efficiencies of three unique AI schemes against the constraint of limited information of the game map. The goal of this research paper, again produced by the likes of well-regarded scholars Spronck and van der Herik, is to test user-developed AI schemes, one of which is modified by the researchers to compensate for the limited map data acquisition, in order to see how they can react to a human limitation of game play: that we cannot know of all the information a map or game holds. Through a quite robust set of experiments, measured in the hundreds of games simulated and played by the AI schemes and including complete measurements of error, weighted and average successes, the research team produced promising results. Even though the researchers acknowledge the limitation of imperfect information on game AI, the game difficulty drastically fell when such a limitation on the AI was introduced, this research still demonstrated an artificial

intelligence system that acts far closer to a human player, which is an invaluable lesson in machine learning for us.

Nareyek, A. (2007). Game AI is Dead. Long Live Game AI! *IEEE Intelligent Systems* (ed. Preece, A.), 22(1), 9-11.

A pithy, witty article in one of the most well-respected magazines concerning the topic of artificial intelligence, decorated scholar and academic writer Alexander Nareyek surveys the state of affairs in AI research, development and demand. Though he has superior credentials, this is not a robust academic work, but rather an informative yet broad overview of the most promising trends for immersive virtual environments in modern games. Noting widely discussed ideas such as automated and interactive storytelling, virtual actors, and reactive, adaptive AI systems, Nareyek at once challenges the reader to think about the possibilities and implications of such new technologies as well as questions the reader on the limits of said developments.

Ponsen, M., Muñoz-Avila, H., Spronck, P., & Aha, D. W. (2006). Automatically Generating Game Tactics with Evolutionary Learning. *AI Magazine*, 27(3), 75-84.

This paper comes from AI Magazine, a scholarly journal released quarterly by the Association for the Advancement of Artificial Intelligence (AAAI). In particular, Pieter Spronck is a Ph.D who has published a plethora of papers on Artificial Intelligence especially in terms of its applications in game worlds. Ponsen and Muñoz-Avila were working with Spronck under a grant from DARPA. The work is relatively current, having been released in 2006, and is pertinent to our work at hand. The audience is other scholars within the field of artificial intelligence, and the article attempts to provide all data about the research they undertook. Their methodology and its results are well documented and their conclusion stands true in the face of their results. This article lays out a clear methodology towards the goal of generating an AI that learns through repetition. Furthermore, the paper established that this evolutionary learning results in an AI that performs better at its task than statically programmed AIs. It applies to our work by giving us a good look at some of the methodology used to actually generate dynamic scripts, especially in terms of the weighted finite state machine they use. Additionally, the work posits that this approach is non-convergent, and in fact, that convergence is a behavior they could not prove occurs. It gives us critical guidance on what methodologies would best suit our “Heartsong” decision engine.

Bakkes, S., & Spronck, P. (2006). Gathering and utilizing domain knowledge in commercial computer games. *Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence*, 35-42.

Bakkes and Spronck work at the Institute for Knowledge and Agent Technology’s Game and AI group; Bakkes being a doctoral student and Spronck a Ph.D and very well

published in this field. This work is also as recent as most of Spronck's other, pertinent work, and it documents the issues with regarding generating domain data and then using that domain data successfully to facilitate learning Artificial Intelligence. Their audience is other scholars in the field of artificial intelligence and they present their experiment with a detailed methodology and mathematical foundation in order to establish that such a system is functionally possible and works. They conclude that it does work, although certain expectations about the relative success of certain algorithms surprised them; they established that it did work and that after increasing repetitions their AI NPCs did improve as their domain state grew. The paper was interesting and established some key questions about what data we need to gather in our DNPC domains and how we would go about utilizing that data. Unfortunately, as with other Spronck papers, although very pertinent and the most applicable to our project, the scope of the paper considers game environments with far more predetermined rule-sets. However, in our implementation of a kingdom with DNPCs, there are not so many or so simple set functions for determining victory or failure, or the concept of progress.