Efficiency and Effectiveness of Game AI

Bob van der Putten and **Arno Kamphuis**
Center for Advanced Gaming and Simulation, Utrecht University
Padualaan 14, 3584 CH Utrecht, The Netherlands

Abstract

In this paper we try to determine the effectiveness of different AI techniques used in simple games. This effectiveness is measured by comparing game-play experience to implementation effort. Game-play experience is measured by letting a test panel play with the different kinds of AI techniques after which a questionnaire is filled in and the implementation effort is simply logged. The results showed that the increasing numbers of AI features is valued, but only until a certain level.

Introduction

Where until recent past graphics was the number one priority when it came to creating games, nowadays we see a shift toward another field as well. As processors get faster, more computing time can be used to create more advanced AI than before [4, 5]. A well applied AI can result in enhanced game-play, a higher replay value and overall a bigger challenge for the gamer.

Specifically this last aspect is where things can go wrong, because one can ask the question if smarter always equals better. Overdeveloped AI can result in games which are too complicated for the core target audience, resulting in negative experiences and thus wasted development time. At the same time, we do not want to create games which lack a decent level of AI, resulting in unchallenging, boring game-play.

Developing a good AI for a game requires quite a lot of development time and resources. Preventing over-development is very important. Putting too much work into the development of AI might not only result in a game that is not fun, but also results in wasted programming effort. In this paper, we try to find a balance between the development effort on one hand and the game experience on the other. In this study, development effort is measured in programming hours and the game experience is measured by questioning a test panel who has played the different games. As this balance will strongly depend on the genre of the game, we limit our research to the field of arcade-style games targeted toward people from 12 to 60 years of age.

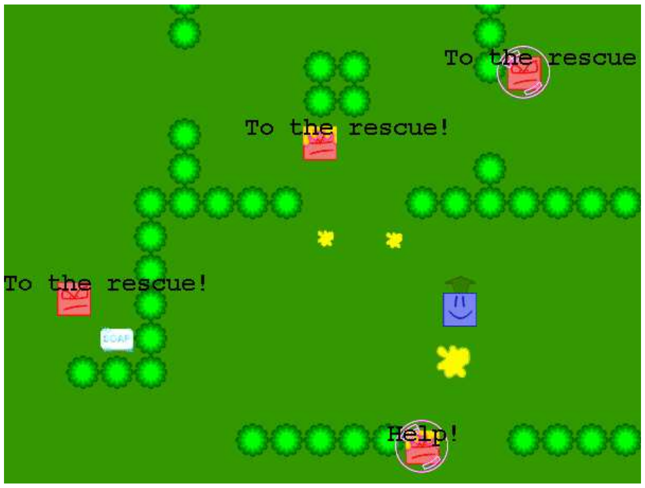

Figure 1: An Agent that will come to the aid of his comrade quickly seems intelligent.

Different AI Techniques

In this section an overview will be given of some of the techniques that are mainly used in games nowadays. Such an overview could never be complete, but it will at least give the reader an idea of the research domain. Please also note that on every technique countless variations can be made, but we have chosen only to treat the common version.

Hard coded Reactive behavior This technique is mainly used by amateur developers as it is easy to implement, but it is very limited in its use. Looking at the current state of an agent and a current event, a simple hard-coded switch statement is consulted to find the new state. This technique has a low complexity, but results in simple reactive behavior which is easy to see through.

FSM Rule-Based behavior With this technique, a Finite State Machine (FSM) is used to create Rule-Based behavior. Again, looking at the current state of an agent and a current event, a state is found. Only this time, a FSM is used, resulting in easy expandability of states and transitions. Per event, the agent now has a number of states he can transit into. Which state this will be is chosen at random.

Dynamic Scripting based FSM The previous technique can be improved greatly by combining it with Dynamic Scripting [7, 8], resulting in learning behavior of the agents. Per event, the agent has again a number of states he can transit into. A weight value determines the chance a particular state is chosen. The weight values can be adjusted in such a manner that the weights of the more efficient responses become higher and thus will be chosen more often by the agents. During the entire game these weights will keep changing so the agents will always keep adapting to the players' behavior.

Machine Learning Machine learning is widely used for finding optimal strategies in competitive domains, i.e. find the strategy that results in the highest possible payoff for an agent. There are several different algorithms types such as supervised learning, unsupervised learning and reinforcement learning. In the application area of computer games, machine learning can be applied at different stages of the development of computer games or during game play. However, machine learning techniques have not been used much in computer games, mostly restricted to non-complex video games [2, 6].

Experimental set up

In order to test the efficiency of an AI technique, it will first be implemented in a small game to log the implementation effort. Then the game-play experience needs to be researched by letting a test panel play the game and fill in questionnaires. The efficiency of a particular technique will be determined using the questionnaires combined with the registered implementation effort.

Outline of the paper

This paper is organized as follows. First, a description of the different AI techniques that were chosen in this research is given. Second, we outline the set up and implementation. This is followed by the description of the experiments and the results. Finally, we give our conclusions and views on future work.

Implemented Game & AI techniques

As mentioned in the introduction, for this research we focus on arcade-style games. Mat Buckland explains in [1]:

> "When designing AI, always bear in mind that the sophistication of your game agent should be proportional to its life span."

Common sense also tells us that when dealing with an arcade-style game, a great number of (sophisticated) AI techniques can be ignored. As we want to determine the efficiency of different techniques, the techniques should differ in expected implementation effort. All techniques based on Machine Learning are not taken into account, mainly because they mostly require extensive learning phases or require on-line learning during game play. The first argument, requiring a long learning phase, makes machine learning not very useful in our test since it is not clear whether this learning time should be added to the development time or not. The second argument, on-line learning during game play, makes machine learning not useful since we only allow players to play the game for relatively short periods of time.

Paint Arena 2D

A simple game, Paint Arena 2D [9], was made (Fig. 1 and 2 were taken from the game.) which includes the different techniques. This is a paintball game in which the player takes on 4 enemies in a small arena. The traditional health bar is replaced by the character itself as it becomes more and more dirty from being hit. The player can take 5 hits before being reset to the centre of the arena. When an agent is hit 5 times, it will respawn at one of the spawn points in the arena. The player can grab soap to clean himself up or use a soap bubble as a protective shield. Agents will not search for power ups to save implementation time. To compensate, they will regenerate health over time and regain a new shield 10 seconds after usage. The different states an agent can be in can be seen in Table 1.

The next three sections will provide the details on the chosen AI techniques:

Simple Reactive behavior

General Being the least complex technique, this technique demands: 1) very short development time and 2) result in a playable game.

The idea of this technique (from now on also referred to as *Tech A*) is that the agent always reacts to the same events in the same way and has a limited amount of actions it can make (Rule base), resulting in behavior which is predictable, but still entertaining. Using this technique, the agent will patrol the area until the player is spotted. It will then switch to his offensive state, until the player is either taken down, is out of range or goes out of sight for a while.

Rule Base As mentioned above, the rule base is hard coded with this technique. This is done purely to save time as this technique should have the shortest possible development time. Below you will find some examples of this technique's rule base.

- If the player comes within 10 yards, I will run towards him whilst shooting (Attack mode).
- If I make physical contact with the player, I will hit him.
- If the player runs away while I am in attack mode, I will follow.
- If the player is outside my 20 yard radius while I am in attack mode, I start patrolling.

Rule-based behavior using a FSM

General This technique builds in theory on Tech A, but now the underlying structure is that of the FSM, resulting in easy expandable behavior. That is why, when using this technique (from now on also referred to as *Tech B*), the agent will be able to react in a different way on the same kind of events.

Rule Base The key of this technique is that an agent can react in multiple ways on the same event and that it will do this randomly. We can't call this random picking of a new state a decision, as it is purely a dice that determines the new state instead of the agent. One event has multiple states attached to it, of which a couple can be seen below:

- If the player comes within 10 yards, I will run towards him (Attack Mode).
- If the player comes within 10 yards, I will run away (Flee Mode).
- If the player comes within 10 yards, I will shoot him (Attack Long Range Mode).

The agent shall randomly picks one of these state transitions. The probability a particular new state is chosen will depend on a weight which is applied to every unique state transition. This weight is user-specified (by the game designer) and fixed during run-time of the game. A unique state transition is identified by its old state, the event and its new state, so each weight has a key which is build out of three components.

The improvement of Tech B over Tech A is that the agents will behave more dynamic and random.

Learning behavior using Dynamic Scripting

General Again, this technique (from now on also referred to as *Tech C*) builds upon the previous technique. This time however, the agents will develop a collective memory, resulting in adaptive behavior of the agents. This memory is realized by letting the agents adjust the weights discussed at Tech B, effectively changing the chance a particular new state is chosen.

As mentioned, each unique transition has a key which is build out of three components: The old state, the event and its new state. We call unique transitions *family members*, if they have the same old state and event in their key. The weights of all transitions in a family should always add up to 100

Mathematical functions are used to determine the effectiveness and the new weight of a particular transition.

Definition 1 (Effectiveness transition)**.** *The effectiveness of a particular transition $E(t)$ is the amount of damage d inflicted on the player plus the received damage r divided by a constant C.*

$$E(t) = d + \frac{r}{C} \tag{1}$$

Definition 2 (Transition weight)**.** *The probability $P(t)$ a particular transition is chosen, is e to the power of the effectiveness $E(t)$, divided by the sum of all the probabilities of the n transitions in this transition family.*

$$P(t) = \frac{e^{E(t)}}{\sum_{i=1}^{n} e^{E(t_i)}} \tag{2}$$

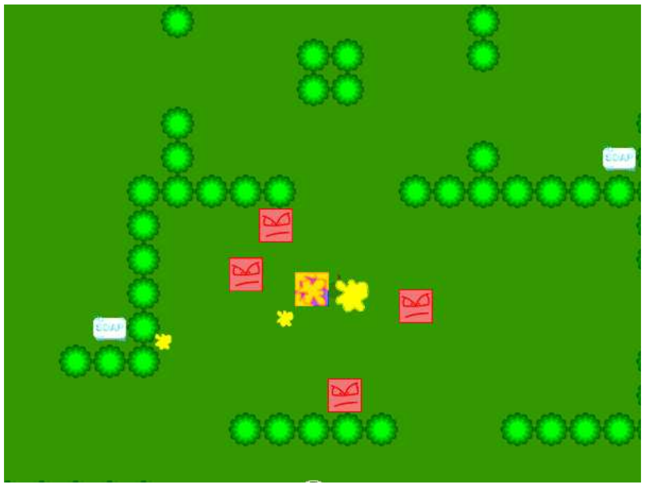

Figure 2: When all agents respond to a help call, things get difficult.

The improvement of Tech C over Tech B is that the agents will keep adapting their behavior to that of the player during the entire game.

Choices made

These particular three techniques were chosen because they are not too difficult to implement but are still very diverse in implementation. Because the addition of states when using Tech A's implementation is very time-consuming, Tech B and C allow for a larger rule-base (much faster addition of states possible because of code architecture). That is why the rule-bases are not entirely equal. Also, almost no time was reserved for tweaking and tuning all techniques. While normally this is a very important part of the process, when comparing techniques this is not so much the case and it was preferred to keep the development times down.

Expectations

The efficiency of the three techniques have to be determined. As mentioned, this efficiency is measured by both programming effort and the game experience.

Definition 3 (Game Experience)**.** *A game provides a good game experience, when the game-play is both challenging, intuitive and unpredictable in such a manner, that the player would very much like to play the game again.*

A number of key elements can be extracted from definition 3, on which expectations can be based. Firstly, there are the hypotheses on the effects of the techniques. These hypotheses describe what influence the techniques have on how the player will play the game.

Hypothesis 1. *With any technique, the player needs to time his actions and react accordingly.*

Hypothesis 2. *The expected improvement by using Rule-based behavior using a FSM over Simple Reactive behavior is that the agents will behave more dynamic, random and therefore are less predictable.*

Hypothesis 3. *It is expected that if the player notices the learning behavior in Learning behavior using Dynamic Scripting, he will try to manipulate the agents.*

Secondly, we can formulate three main hypotheses on both the influence of how the player needs to react on the actual game-play and the game experience of the player.

Hypothesis 4. *If the player needs to time his actions and react accordingly, the game-play will be improved.*

Hypothesis 5. *When agents behave more dynamic and less predictable, the game-play will be improved.*

Hypothesis 6. *If the player is able to manipulate agents, the game-play and replay value will be improved.*

Hypotheses 1 and 2 state that if the AI technique becomes more complicated the player is pushed to adapt more. Hypotheses 4 to 6 state that if the player needs to act more to defeat the AI, the game play is improved.

Set Up and Implementation

Set Up

In order to test the efficiency of the chosen AI techniques, the techniques themselves were implemented. To this end, a game will be made which implements the AI techniques and the development time of the three different AI techniques will be logged.

With the programming effort known, only the game-play experience needs to be researched. To this end, a questionnaire should be developed and a test panel is needed to play the different types of the game and fill in the questionnaires.

The efficiency of a particular technique can then be extracted from the filled in questionnaires combined with the registered implementation effort.

Implementation

As mentioned before, Paint Arena 2D [9] was made to implement the different techniques.

When the game starts, a flood fill algorithm automatically fills the level with a navigation graph. This graph is used by the agents for patrolling (A* search to random node), attacking, assisting and pursuit (A* search to the player) and fleeing (A* to level corner away from the player). As the agents always stay on this graph, no collision detection against the level is needed.

Tech A When using this technique, the agent is able to get triggered (e.g. when the player comes near), resulting in a state switch. Note, however that, to save time, a FSM isn't used, but instead the different events are hard coded in one big switch statement. The only states implemented here are Patrol and AtackLR. This results in a very simple game with a short development time.

Tech B For this technique, a FSM is used, following Mat Buckland's example in [1]. All states from Table 1 are implemented and the transition weights are chosen by insight.

State	Description
Patrol	Wander through the arena
Call Help	Call the help of the other agents nearby
Run to Player	Run towards player position until in short range
Attack LR	Shoot with (weaker) long range weapon
Attack SR	Shoot with (stronger) short range weapon
Shield	Place a soap bubble around yourself
Hide	Run for a corner of the arena, away from the player
Assist Agent	Run towards player position until in long range

Table 1: States an agent can be in

Tech C The same FSM from Tech B is used, but this time an extra feature was added that monitors the effectiveness of decisions of the agents.

Every time the changeState method of the FSM is called by an agent, this state change (also referred to as a *decision*) is stored to a special class named *TransControl*. With this *decision* is stored: the current time, a pointer to this agent, the old state, the condition and the new state.

The three parameters ”old state”, ”condition” and ”new state”, form a combined primary key of a transition, from now on referred to as an *unique transition*. The collection of all *unique transitions* of which the first two key parts are the same, is referred to as a *family*.

When the game starts, the weights equal those of Tech B. Every iteration of the games main loop, *TransControl* is checked to see if decisions made in the past can be judged yet. To this end, the time stored with the decision is checked with the current time. If a predefined amount of time has passed, the effectiveness of the decision is calculated using Equation 1 and the weights within the *family* of this decision are updated using Equation 2 for every family member. An addition needs to be made for ”Call Help”, as the damage done by assisting agents must also be taken into account when calculating effectiveness. Therefore a list of *assists* is maintained, storing the assisting agent's ID, the victim agent's ID and the current time as soon as an agent goes into the ”Assist Agent” state.

If the agent dies before his *decision* is judged, the *decision* is discarded as the amount of damage taken can't be determined anymore. If the player died before a *decision* was judged, the maximum health of the player is added to the inflicted damage to prevent wrap-around negative numbers. E.g. a player with 2 health taking 4 damage ends with 3 health, as it died and respawned with 5 health. This would give a score of -1. When 5 is added, we get the desired score of 4.

Experiments and Results

To test the effectiveness of a technique, both the implementation effort and the game play experience is needed. The implementation effort is maintained during development and can be found in Table 2. Experiments are needed to deter-

AI Technique	Hours
Tech A	31
Tech B	45
Tech C	87

Table 2: Programming effort (By one person)

mine the game play experience. We conducted an experiment using a test panel to determine this experience.

User tests

A questionnaire was developed to test the experiences of the test panel, holding questions about fun factor, difficulty, predictability, replay value and comparison questions between the played versions. An event logger was also implemented in the test game, logging facts like score, number of deaths, shielding, number of kills and long and short range bullets fired.

The test panel consisted of 25 testers, ranging from programmers to house wives and from 20 to 60 years old. Subgroups (4 to 5 testers each) were created in such a way that testers from the same category (e.g. programmers) were spread out as much as possible.

Hypothesizes 1 to 6 amongst others were used to create the user tests themselves.

Experiments

The order in which the versions are presented to the test panel will influence the outcome, as people become more experienced after playing the game. To negate this effect, we divided our test panel in 6 subgroups, each only comparing two versions. This resulted in groups testing AB, BA, BC, CB, AC and CA.

Each subgroup first gets to play the first version for 5 minutes, has a 30 seconds break before testing the second version for 5 minutes, after which the questionnaire must be filled in directly. Because another subgroup tests the same two versions in opposite order, we minimize the influence of the order of presentation. A practice level was added so the tester can familiarize with the controls before starting the actual test.

Results

As mentioned, Hypotheses 1 to 6 amongst others were used to create the user tests. It are these Hypotheses that are used to formalize the results.

Results entire test panel These are the combined results from all test panel members. Table 3 shows in the second column that Hypothesis 1 is confirmed. Strange enough, with Tech B and C the player believes less timing and reaction is needed. The third column states that when the AI gets more advanced, the player feels he can manipulate the agents more. While in fact, this was only possible with Tech C.

The perception of the agent's behavior by the player is stated in Table 4. It shows the tester's opinion on agent behavior. E.g. 75 percent of the testers who played with Tech A, found the agents more predictable than with the other techniques. Tech B is both more believable and more intuitive than the other two. This outcome confirms Hypothesis 2. The large difference between Tech B and C is remarkable.

Tech	Timing	Manipulate agent
A	100%	6%
B	66%	33%
C	81%	44%

Table 3: Outcome questionnaire on Hypotheses 1 and 3. Percentage of testers that claimed to have made certain actions.

Behavior	% A	% B	% C
Believable	31	66	43
Intuitive	38	66	31
Predictable	75	50	19

Table 4: Outcome questionnaire on Hypothesis 2. The percentage of testers that played a particular AI version and found that version strongest in a particular genre of behavior.

Tech	Score	St.Dev.
A	3.40	1.05
B	3.41	1.33
C	3.67	1.05

Table 5: Replay value. Average score on a 1 to 5 scale.

Posing	% A	% B	% C
More actions needed	31	50	50
Shorter playtime	44	44	31
More fun game	*38*	*56*	*50*

Table 6: Outcome challenge and fun of the game. The percentage of testers that played a particular AI version and found the posing most applicable on that version.

As for Hypotheses 4 to 6, the game-play needs to be evaluated. Therefore we questioned the players about the key elements from Definition 3. In Table 5 can be seen that the difference in replay values is negligible. As for challenging, Table 6 shows that when playing with Tech A, only 31 percent of the gamers claimed they needed to perform more actions for the same result with this technique. This indicates that the larger part had less trouble with this technique, resulting in Tech B and C seeming more challenging. The difference in time perception here is negligible.

The results of the following statements provide a clear answer to Hypotheses 4 to 6:

- As the game demands more timed actions and reacting, I enjoy the game experience more. *Score: 79%, st. dev 0.76*
- As the enemy becomes less predictable, I enjoy the game experience more. *Score: 83%, st. dev 0.69*
- If I can manipulate the enemies' behavior with my ac-

tions, I enjoy the game experience more. *Score: 83%, st. dev 0.83*

Finally, the gamers claim in Table 6 that playing with Tech B was most fun.

Conclusion

Despite the fact that the difference in techniques is recognized, the gamers do not seem to have a particular preference. Especially the differences in replay and agent behavior is hardly noticeable. However, on predictability and overall judgment is Tech B preferred over the other two techniques.

Looking at Table 2, it is clear that with increasing features, the implementation effort increases as well. It shows that Tech C costs twice as much effort to implement than Tech B. As the results show that Tech C does not score twice as much as Tech B, it can be concluded that Tech B is more efficient than Tech C. The difference in efficiency between Tech A and B is not so obvious. As Tech B scores higher on game-play experience, but at the same time costs more implementation effort, Tech A and B could be called equally efficient. As Tech A and B are equally efficient, but Tech B is valued more, Tech B is preferred in this context.

As the players state that the game experience is improved as the enemy becomes less predictable, it is advised to make larger rule bases and create good algorithms for random decision making. When making simple games this is preferred over putting more effort in the AI system itself.

This research was limited to arcade-style game-play and aimed on some techniques which are easy to implement. When looking at other game genres, the AI techniques used are very different. Lots of research can be aimed towards these other, more advanced genres, including more difficult AI techniques [3].

Acknowledgement

The authors would like to thank Marco Wiering and Rob van Gulik for their continuous support during the development of the test game. The authors would also like to thank Dennis Nieuwenhuisen for the use of the Atlas library for graph handling. The research was partly supported by the GATE project and funded by the Dutch Organization for Scientific Research (N.W.O.) and the Dutch ICT Research and Innovation Authority (ICT Regie).

References

[1] Buckland, M. 2005. Programming Game AI by Example. Wordware publishing, inc.

[2] Caruana, R. 1997. Machine learning. Multitask Learning 28:41–75.

[3] Caruana, R. 2001. The future of game ai: A personal view. Game Developer Magazine 8:46–49.

[4] Champandard, A. 2004. AI Game Development. New Riders, Indianapolis, IN.

[5] Funge, J. 2004. Artificial Intelligence for Computer Games: An Introduction. A K Peters, Ltd.

[6] Palmer, N. Machine learning in games development. `http://ai-depot.com/GameAI/Learning.html`.

[7] P.H.M. Spronck, I.G. Sprinkhuizen-Kuyper, E. P. 2003. Online adaptation of game opponent ai in simulation and in practice. In Q. Mehdi, N. G., ed., GAME-ON 2003, Proceedings, 93–100. Ghent, Belgium: EUROSIS.

[8] P.H.M. Spronck, I.G. Sprinkhuizen-Kuyper, E. P. 2004. Enhancing the performance of dynamic scripting in computer games. In Rauterberg, M., ed., ICEC 2004, lecture notes in Comp Science 3166, 296–307. Berlin, Germany: Springer-Verlag.

[9] van der Putten, B. 2007. Paint arena 2d on www.codedfun.com.