23

Personality Reinforced Search for Mobile Strategy Games

Miguel A. Nieves

23.1 Introduction

The design philosophy central to the critical acclaim and success of the strategy war game *Battle of the Bulge* (iOS) was that every move can be used to express personality as well as skill. Despite being based on a board game, determining how the AI should play proved to be quite complex. The plethora of rules, units, map characteristics, and nondeterministic outcomes led to an explosion of game states. Since the game was new, no established strategies existed. This, coupled with the processing and memory limitations of mobile development, made searching to a win state impossible.

Too often, a win state is the focus of development and "personality" is expressed using a random range of positive, but suboptimal choices. In an attempt to win, some choose to skew less visible odds in the computer's favor. Some developers navigate a quagmire of numeric optimization of their state evaluator. These approaches are fundamentally attempting to solve the wrong problem. It is not "How do I win?", but "How do I play?"

Freed of the focus to win, other decision-making metrics can be explored. These data-centric methods are mixed with traditional tree-search approaches to create a personality-centric informed search. The overall architecture is surprisingly similar to the triune model of human brain evolution. This model combines instinct, experience, and prediction in a simple and intuitive way that supports agile development methodologies.
This chapter will describe how we:

- Used the triune brain model as a guide to perform a personality-based tree search
- Created visible development goals, enabling greater stakeholder confidence
- Started making decisions quickly using utility for personality-based decisions
- Simplified game state for faster iteration
- Created adaptive behavior that doesn't assume an opponent personality
- Removed hand tuning by allowing the AI to breed
- Understood how personality can be expressed via optimization

While this chapter's focus is on optimizing for personality in turn-based strategy games, these techniques are applicable to problems with a large number of potential states and no clear dominant strategy.

23.2 Turn-Based Strategy War Games Based on "Euro" Board Games

Euro-style board war games mix strategy, chance, and complex rules to keep wins close, but just out of reach until the very end. This makes them very engaging but difficult to quickly score or evaluate for equivalency. Like many board games, *Battle of the Bulge* is played by moving pieces on a map, except that each piece and map location are associated with detailed stats and rules. Units may also attack each other, but combat is nondeterministic. Thus, a "good" move can still fail and cause a player to reevaluate their plans. Determining victory is often multifaceted and can occur instantly due to accumulated points and positioning or can occur after a fixed number of turns.

The general flow of a game is as follows:

Day Start → Some Number of Player Turns → Day End Rules/Victory? → Next Day Start

Each player turns consists of the following:

Turn Start → Special Rules Decision → Movement Decision → Combat(s) → Movement after Combat Decision → Turn End → Next Player Turn

There are many factors that make the analysis of this game complex. Player turns do not always alternate, but the opposing player cannot move until the first player has finished. Movement rarely involves a single game piece or unit. For each unit, there are many potential destinations, multiple paths, and even side effects from moving at all. Sometimes, additional movement decisions are needed after combat. Additionally, many action permutations yield the same outcome. This results in a huge branching factor even before exploring enemy counter moves!

Before, during, and after each move, there are numerous, potentially dynamic, rules that must be referenced or cross-referenced to determine legality. Every space may contain both friendly and enemy units and even those units have changing statistics. Past actions may influence the turn flow. These factors all contribute to a game state that is expensive to store, copy, and compare for equivalence.

Not being able to reach a significant terminal state puts a lot of pressure on the state evaluator. An overworked state evaluator is difficult to maintain and becomes even more complex when game designs are asymmetric. Without significant time spent studying and playing, what constitutes a good move remains subjective. Designers rarely have the answer, even for their own game system. Most often, their answers require expensive analysis, further slowing an already laden system.

Perhaps, there are some lucky developers who do not have to answer to timelines, CPU restrictions, or dwindling budgets, but that is not our situation. Our stakeholders need quantitative results to justify the budget we request of them. These are not villains, but real people who feel betrayed when their AI expert doesn't deliver something up to their undefined metric of "good." To them, it doesn't matter how little processing power a user's device has or how many times the game designers changed the rules out from under the AI. What matters is that a challenging game is released, on time, and is profitable enough that everyone can still keep doing what they love. Fortunately, there is a solution, and one that looks shocking like… *the human brain*! Don't laugh… okay, laugh a little bit, but bear with me and you'll see how remixing traditional techniques and optimizations for mobile come together to create a smarter personality-driven opponent.

23.3 Evolution and the Triune Brain

The triune brain model posits three brains emerged in succession during the course of evolution and are now nested one inside the other in the modern human brain: a "reptilian brain" in charge of survival functions, a "limbic brain" in charge of memories and emotions, and the "neocortex," in charge of abstract, rational thinking [Dubac 14]. While this model has been criticized as an oversimplification in the field of neuroscience [Smith 10], it makes an interesting metaphor in creating our personality-driven architecture.

We build these three brains, shown in Figure 23.1, into a loosely coupled, tunable, informed search algorithm capable of expressing personality as well as skill. The number of moves, permutations, and game states war games generate make it essential to find ways to only explore the moves that matter. A good start is to use a search algorithm like alpha–beta, for improved tree pruning over minimax. However, more aggressive methods are needed to prune the tree to one that can be explored by mobile hardware. An interpretation of Occam's razor is that if there is a good reason to make an assumption, then do so even if is not always true [Birmingham 77].

This is where our other two brains can provide information on what it thinks is important and what the player thinks is important. Our reptile brain provides move ordering via tunable utility-based heuristics. The limbic brain records and recalls player metrics gathered from other play sessions to understand the past. And the search function in the neocortex brings all this information together to determine, through the lens of the state evaluator, likely counter moves.

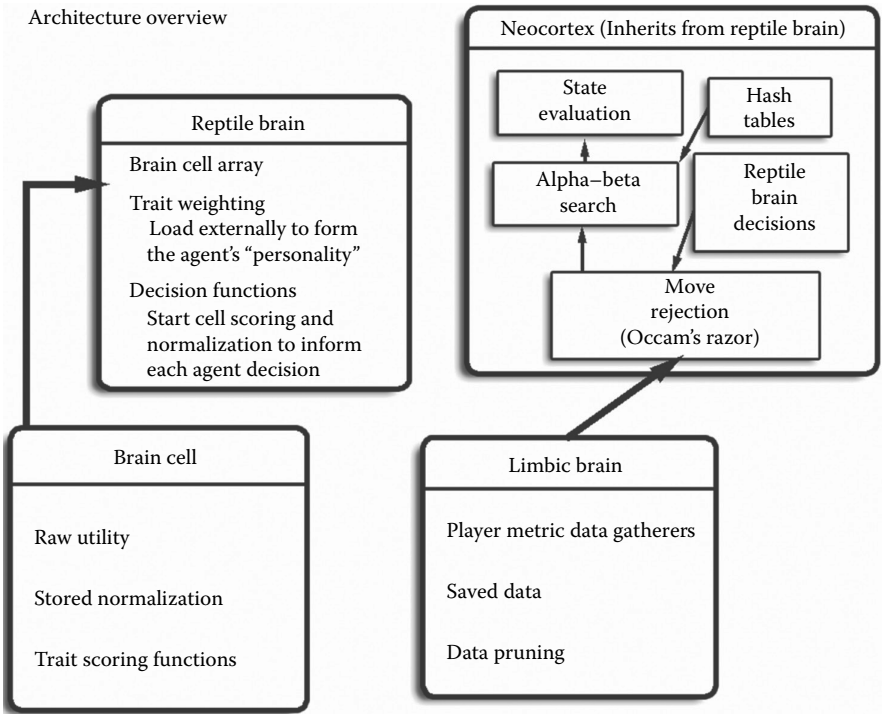

Figure 23.1

Architecture overview for triune brain–modeled AI.

23.4 Personality First

Many developers try to create the perfect player first, then down tune to create personality. This prioritization discounts the great wealth of expression players have on each of their chosen battlefields. Within every game, there are very different ways to play. Thinking about opponents you've personally faced in games, what makes Mary different from Sue is how she might approach the problem, what she thinks is important, and her temperament during play. What sort of characteristics make up how they choose moves? Does she carefully consider each move or brashly jump on any potentially victorious outcome? What does she consider a win? Building in these metagame aspects can help create AI opponents that "feel" like human players. Play becomes richer because strategies are formed not just within the game rules, but in how another will react to those rules.

The personality in the triune architecture comes from the reptile brain. In this brain, we create utility functions that explore only "one" aspect of play. Think of what a player considers before a move. "I'd be a real jerk if I moved here" is a valid heuristic. We look for these factors in the same way you'd score the space that would yield the highest point gain. The goal is to find the move that exemplifies that single aspect. These decisions may or may not take in account counter moves; the goal is to make each determination in near linear time.

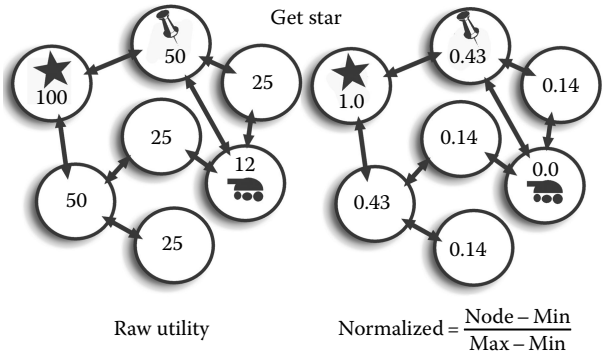

Figure 23.2

Sample output for the "get star" trait.

In the following examples, each circle represents a space that can be moved to. Arrows indicate how spaces are connected. The tank is the player's location and the other graphics are features of the space. Raw utility is a number system we have made up to help quantify the world.

Consider Figure 23.2. If we only care about the star, then what are the best spaces to be in? These are evaluated with raw and normalized utility. Now, what if we do not want to be near a tack? States are evaluated with this metric in Figure 23.3. We now have two utility functions, one for moving toward the star and another for avoiding a tack.

Here, we've created two *wildly* different number systems! How do we determine what to do? We use normalization [Mark 08]. There are lots of different ways to express the quality of a space, but normalized values are easy to compare and combine, as shown in Figure 23.4. Best of all, they are easy to scale.

Personality is created in a consistent way when you scale the normalized outcome of each utility function. It also helps if you keep in mind archetypes [Ellinger 08] when building your utility function. Archetypes are bold behaviors that can be described by a single

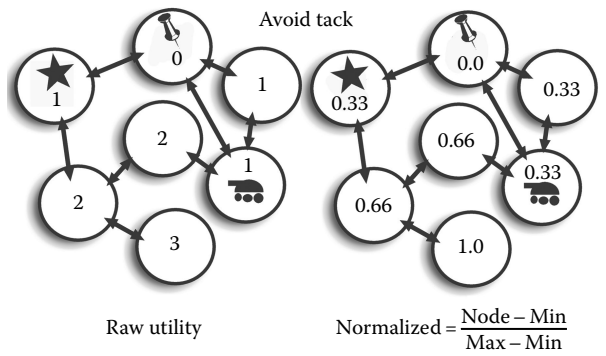

Figure 23.3

Using a different numeric basis for other traits still works when normalized.

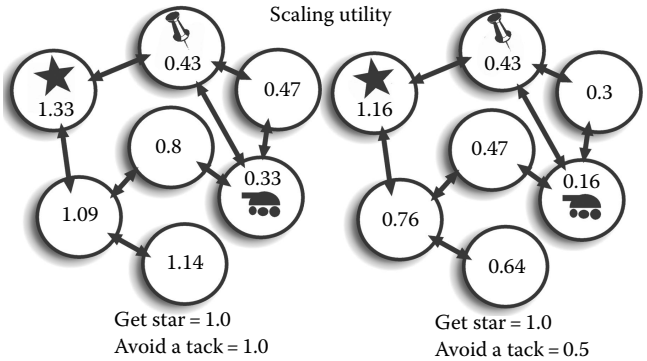

Figure 23.4

Scale and combine normalized values to achieve a unique personality.

word, such as "coward," "defender," or "psycho." While this may use a few different types of data, it still has a singular focus. It may help to think of each of these utility functions as personality traits. Testing each trait in isolation reveals new ways to think about how a player might move through the game. It also presents the chance to think critically about what information is important to interesting decision making.

In adjusting values for different traits, traits can even be scaled by negative numbers. A trait may have a corresponding and interesting opposite. This is because as the traits are combined, the scaling is relative. Scaling a trait A by 5 and trait B by 10 is the same as scaling trait A by 0.2 and trait B by 0.4. Conversely, if you have a several conflicting traits and scale them all the same, the move choice will be similarly conflicted and indecisive. It's very important for archetypes to be clear and obvious.

These hand-tuned values should be loaded externally to start, but as more traits are added, the "test, tweak, repeat" method becomes quite time consuming. AI versus AI play can be used at this point to validate potential agents, but before running off to build our genetic algorithm, we should move on to our neocortex.

23.5 Neocortex

The neocortex is where all the gathered data is used. For now, the development of the limbic brain will not help as much as the gains from early performance testing and more competitive play. The neocortex is the most traditional portion of this architecture. It starts with a lightly optimized implementation of alpha–beta search. For now, this should be kept independent of the reptile brain. The key features needed early on is the ability to control the number of plies (turns) explored and a set of counters to help determine which decisions get made the most. For now, the only optimizations focus on greatly reduce processing time. When optimizing any tree-search algorithm, the major hot spots are likely to be

- Legal move generation
- Path finding
- Copying game state
- The state evaluator

With a game state as complex as required by war games, transposition tables are your best friends. Initially, this may seem daunting, but these lookup tables don't have to be complete. We create them so they capture only enough information to determine if duplicate processing is about to happen. If units A and B want to go to space Z, unit A could move first then B, or B first then A. The end result is an identical state of "AB in Z" or "BA in Z." That's not so bad, but if we only looked at one of those states, we'd be twice as fast. What if there are three units in space:

"ABC in Z," "ACB in Z," "BAC in Z," "BCA in Z," "CAB in Z." What if you also were exploring Space Y?

The trick to getting this working for complex games turned out to be surprisingly simple. A string much like the aforementioned examples is generated and hashed. This creates a unique number and is added to other hashes to create a final master hash. So the string "unit A to Z" might create a hash of 42 and "unit B to Z" might create a hash of 382561. Add the two numbers together for the final move's hash. If the combination comes up again, the code knows not to reexplore that branch. Care must be taken if XOR is used to combine these hashes. If two units have the same value and move to the same space, they will XOR to 0. This can cause false positives (hash collisions) that would be difficult to identify.

Lookup tables can be exploited when dealing with legal move generation as well. In our case, we had three units that could move during a turn, but the first unit could invalidate a move by the second. It was much faster to store the initial legal moves and then check if a previous move would make a current move illegal. This type of lookup table also speeds up path finding. By storing the best path found, expensive path creation only happens once. Using the ply count as our index lets us keep tables small and searches fast.

Simplifying data became essential when attempting to keep track of game states. Initially, we experimented with copying the entire state at every decision point. Copies are made so we can speculate along a decision branch and perfectly return to that point and to try a new decision. With each game having hundreds of pieces, each with multiple mutable attributes, it was expensive to copy all states (unlike a chess board of 64 integers). Since alpha–beta is a depth-first method of searching, we didn't feel memory pressure despite the massive size of state, but the system struggled as it processed tens of thousands of copies. This only seemed to get worse as games progressed.

The first step was removing as many copies as possible by tracking and unwinding the changes using the same mechanisms developed for player level undo. This drastically reduced the copies, but it did not remedy the late game slow down. Some planning algorithms reduce their game state to (action, result) pairs. Building on this concept, the only data copied with the final game state was the action list changes from the initial decision point and the score provided by the state evaluator. In the late game, this took the number of actions copied per state from several thousand to a few dozen. The initial copy of game state is also simplified down to only what is needed for rejecting, applying, and scoring future moves.

23.6 State Evaluation

The neocortex's state evaluation is focused on raw numeric values. Even with asymmetric games, it is initially important to keep state scoring as symmetric as possible. Numbers like army health (normalized), spaces owned, special rules spaces owned, points scored,

and if the state is a win, loss, or draw are a good place to start. Each unit's distance from the goal or enemy line is good for keeping units moving. If any strategies are revealed during the reptile brain tests that are easily quantifiable, include those as well (e.g., unit type A is more likely to win in combat if unit type B is also present). It is important to keep the state evaluator as close to linear or polynomial time as possible since this will be run several thousand times during search.

The next step is to verify the neocortex in isolation by having it find the best decision for a 1-ply search. This is also a good baseline for performance testing. While you should allocate resources as needed to improve game code, it is important to determine acceptable wait times for decisions. The 1-ply search should be relatively fast and its data can be used to improve the ordering of the moves searched in the multi-ply search.

However, the primary method of optimization is in the application of Occam's razor to the move space. Aggressive move rejection or "chopping" prevents wasting time on useless or redundant moves. This becomes especially important as more plies are processed and impacts all other problem hotspots in our search performance. Consider the size of the searches:

10 Moves for P1 → 10 Counter Moves for P2 = 100 Game States Processed
2 Moves for P1 → 10 Counter Moves for P2 = 20 Game States Processed
2 Moves for P1 → 2 Counter Moves for P2 = 4 Game State's Processed
10 Moves for P1 → 10 Counters for P2 → 10 Counter-Counter Moves for P1 = 1000 States
2 Moves for P1 → 10 Counters for P2 → 2 Counter-Counter Moves for P1 = 40 States
2 Moves for P1 → 2 Counters for P2 → 2 Counter-Counter Moves for P1 = 8 States

Random move chopping provides a baseline for the real-world processing savings, but we can bring intelligence and personality to our move removal. The chopping function itself will be run even more than the state evaluator so it's important not to load it down with too much logic. The chopping function scores each legal move for basic factors like terrain, distance for the origin, and current owner.

At this point, we can finally add the value of the move as decided by the reptile brain! Before kicking off the alpha–beta search, we use the reptile brain to score what it thinks are the best moves. This time, instead of using the legal move range of a unit, we pretend the unit can reach *every* space. We let the reptile brain rank every space then store the normalized value. Finally, we apply a scaling factor in the chopping function that allows the other numeric factors to break ties.

CAUTION: If there is an instant win space, ensure its utility doesn't stomp out the expression of the other spaces. If the win space score is 10,000, but all other spaces score <100, linear normalization puts yields 1.0 and <0.009. Work to keep the final number ranges within reason, to get the most expression from the reptile brain.

As in the examples earlier, the amount you chop can be asymmetric, externally loaded, and even dynamic. Using the reptile brain to guide the chopping function allows its personality to be expressed and converts a downside of chopping to your advantage. If the reptile brain is "aggressive," it will likely remove nonaggressive moves. Unfortunately, this could mean a potential good move could be missed. Also, since we only have a limited

time with our games before they need to ship; it is very likely there is a style of play that we have not accounted for. Fortunately, since our function is dynamic, and our evaluator robust, we can add the final piece of the puzzle.

23.7 Limbic Brain

One of the amazing things about the human brain is our ability to learn. As game developers, one constant is that players will always surprise us. So, we aren't reliant on anecdotal data; we use metrics to learn how players play to improve their experience. In marketing, this might be used to determine which ad experience provides the best return and click through rates. We can use this to build a model of player behavior and guide our agents to interrupt emerging strategies.

Our implementation of the limbic brain is similar to "heat maps" as seen in shooters. Since our games are turn based, the major difference is the use of time. When a player makes a move is just as important as what move that player makes. The data we want to gather should not necessarily be related to previous evaluation functions. We must assume the player has a better idea of what makes a space or move valuable. We record each time a player makes a move or passes through a space. It doesn't matter if the move led to the player's win or loss, the goal is to interrupt an emerging dominant strategy. This data can be used right away, but it is important to scale its effect based on sample size.

The most valuable place for this data is the minimizing player's chopping function, specifically where the AI must guess which move a human opponent will make. Alpha–beta assumes optimal play by both sides. This is less reliable on shallow searches, especially when the quality of the terminal state evaluation is ill defined. Since the inclusion of the reptile brain, the chopping function would have only assumed our opponent played with the same personality. The inclusion of actual player data lets the AI learn from past actions and start to model its opponent.

This model can do more than reduce counter moves. Integrating the limbic brain data back into the reptile brain creates a new heuristic. This heuristic scores where the player moves the most during the next turn. By including ownership/defense of player-valued spaces in the state evaluator, soon the AI appears to use human strategies for itself. We updated the chopping function to always include one player-valued space.

23.8 Evolution and the Genetic Algorithm

Each of these systems can be developed largely independently to start, but as seen with the addition of the limbic brain, they add a great deal of value to each other when used together. As in the triune brain model, these systems help each other and become more interconnected. Ideas from one brain feed solutions to others. These systems can be leveraged to optimize and even produce higher quality results.

Even with personality traits as guides for the reptile brain, it becomes time consuming to hand tune values. This is especially true when looking for "perfect" play or when games take a long time to play. We taught our code to tune itself using a genetic algorithm. The reptile brain already loads values externally, so they can be evaluated through scoring the outcome of a few AI versus AI games. Playing several games is important because it reduces "lucky dice" in games with nondeterministic play. It also gives you a better idea

of the AI's actual abilities. Since the state evaluator gives us more information than win or loss, the final state evaluation gives a good indication of fitness of a strategy. Early on, we only tune one AI at a time so there is a fixed point of comparison. This helps narrow the search space since you are likely to have multiple floating-point values in your AI's chromosome.

23.9 AI Chromosomes

Genetic algorithms work by combining, mutating, and testing a set of values on a problem space. The chromosome contains those values. In our case, a sample chromosome might look like the following:

Aggression = 0.5, LimbicBrainValue = 0.2, SpaceGain = 0.2, Seek_Pie = 1.0
Or encoded, this would be: 0.5|0.2|0.2|1.0.

The key with encoding is to create a flexible enough data structure that makes it easy to import values into the reptile brain. Because this isn't a structure needed for performance, compactness isn't as important as readability and exportability. Dictionaries work well here, and a few small variables let the chromosome keep some information about itself. In our chromosome, we stored the following:

Genes (Dictionary[string, float]) → These values will get changed and mutated.
Best & Worst Score (Float) → Indicators of the range of fitness.
Total games (Integer) → How many games has this chromosome played.
Total Sore (Float) → Combined fitness of all games played.
Average Score (Float) → Could be calculated, but occasionally nice to have around.

There are a few really great tutorials on best settings and trade-offs when creating your genetic algorithm [Buckland 02]. One alteration to mutation that is especially helpful when dealing with floating-point-style chromosomes is to modify instead of entirely replacing a gene. This way, the floating-point value being mutated goes up or down based on a maximum random amount you specify. All gene values should be kept within a certain range to ensure readable data is fed to the reptile brain.

The tiny size of each chromosome makes them easy to keep in memory. This can be leveraged to provide quality debug output of all the chromosomes tested. Early in development, this is essential since AI on AI games will likely expose game-breaking bugs. We saved each generation separately so we could view the propagation of chromosomes and separately saved a master list of all chromosomes sorted by a meaningful metric like highest average fitness. That metric has a tendency to hide big losses with big wins, so a better metric is a winner with the smallest best-to-worst score range.

Reviewing the gene values of winning chromosomes provides further insight and validation into your creation of reptile brain heuristics. In the example earlier, the "Seek_Pie" gene was set to our maximum value of 1.0. As a developer, my love of pie may have blinded me to actual winning strategies. The genetic algorithm, at some point, may mutate a "Seek_Pie" gene to a lower value. If this is the case, it may turn out that these heuristics may need to be reworked or avoided in the future. Conversely, heuristics that may not have seemed as valuable could turn out to be very useful.

23.10 Conclusion

There are many approaches to improving the number and quality of game states searched via alpha–beta or similar search algorithm. Simplifying games states, creating transposition tables, throttling, and move ordering alone does not make for smarter or more interesting opponents. In looking to our own brain's evolution, we have created an AI that can play with personality, plan ahead to the future, and even predict ways to interrupt player strategies. Parallels to our own biology make it easy to explain to stakeholders. It can be evaluated with concrete goals and feature sets. Lastly, it reduces the amount of hand tuning usually required by utility-based systems by leveraging the power of genetic algorithms.

This system has evolved with and powered the AI of *Battle of the Bulge* (reptile brain), *Drive on Moscow* (triune brain), and *Desert Fox* (optimized triune brain). While the game systems are similar, each game has wildly different rules, maps, and units for each scenario. Each game grew in complexity, and the number of game states with it. While not explicitly a general game player, the triune brain system has the flexibility to deal with each new situation without major architecture reworking. By focusing on personality rather than winning, and data collection over code crunching, interesting AI opponents are created even in the face of complex game systems.

References

[Birmingham 77] Birmingham, J.A. and Kent, P. 1977. Tree-searching and tree-pruning techniques. In *Advances in Computer Chess 1*, ed. M.R.B. Clarke, pp. 89–107, Edinburgh University Press, Edinburgh, U.K., ISBN 0-852-24292-1 [reprinted in *Computer Chess Compendium*, D.N.L. Levy (ed.), pp. 123–128, Springer, New York, 1989, ISBN 0-387-91331-9].

[Buckland 02] Buckland, M. 2002. *AI Techniques for Game Programming*. Premier Press Game Development, Cengage Learning PTR. http://www.ai-junkie.com/ga/intro/gat1.html (accessed September 10, 2014).

[Dubac 14] Dubac, B. 2014. The evolutionary layers of the human brain. http://thebrain.mcgill.ca/flash/d/d_05/d_05_cr/d_05_cr_her/d_05_cr_her.html (accessed September 10, 2014).

[Ellinger 08] Ellinger, B. 2008. Artificial personality: A personal approach to AI. In *AI Game Programming Wisdom 4*, S. Rabin, Ed. Charles River Media, Hingham, MA.

[Mark 08] Mark, D. 2008. Multi-axial dynamic threshold Fuzzy Decision Algorithm. In *AI Game Programming Wisdom 4*, S. Rabin, Ed. Charles River Media, Hingham, MA.

[Smith 10] Smith, C. 2010. The triune brain in antiquity: Plato, Aristotle, Erasistratus. *Journal of the History of the Neurosciences*, 19:1–14. doi:10.1080/09647040802601605.