**Overview of approraches to DDA**

##### Probabilistic Methods:

##### Description: Probabilistic methods use statistical approaches to adjust game difficulty. They can involve stochastic decision-making, probability distributions, or even Bayesian methods to model the player’s skill level and adapt accordingly.

##### Advantages: They can handle uncertainty and randomness very well, offering a broad range of adaptive experiences. They are also flexible and can be incorporated with other AI techniques.

##### Disadvantages: Implementing these in Unity can be computationally expensive and might not always offer consistent results. It can also be challenging to fine-tune the probabilities for optimal player engagement.

##### Single and Multilayer Perceptrons:

##### Description: These are simple neural network models, with single-layer perceptrons being the simplest form of a feedforward neural network. Multilayer perceptrons add additional layers of neurons to increase complexity and potential accuracy.

##### Advantages: Both types can handle more complex decision-making and learning scenarios than some traditional AI methods. Multi Layered perceptrons can model more complex behaviors due to their depth.

##### Disadvantages: Implementing neural networks in Unity may require more processing power and may increase the complexity of the development process. Also, the training data can heavily influence the results.

##### Dynamic Scripting:

##### Description: Dynamic scripting is a technique where game AI adjusts its tactics and strategies based on player performance using predefined scripts. Scripts are weighted and selected based on their historical success rates.

##### Advantages: This approach is relatively simple to implement, offers a clear design framework, and can quickly adapt to player skill levels.

##### Disadvantages: In Unity, managing large script libraries can be challenging. It also risks becoming predictable if the script library isn't diverse enough.

##### Hamlet System:

##### Description: The Hamlet system (Human-AI Model Learning from Event Traces) uses a data-driven approach to model player behavior and adjust AI actions accordingly.

##### Advantages: It can provide a more personalized experience by learning from player behaviors and is robust to changes in player strategy.

##### Disadvantages: It can be challenging to implement in Unity due to its complexity and need for extensive player data.

##### Reinforcement Learning:

##### Description: Reinforcement learning is an area of machine learning where an agent learns to make decisions by performing actions and receiving rewards.

##### Advantages: It is an active area of research with promising results, capable of creating AI that can adapt over time and improve its strategies based on feedback.

##### Disadvantages: Implementing reinforcement learning in Unity can be complex and computationally expensive. It may require a lot of training data and time to converge to a useful policy.

##### Upper Confidence Bound for Trees (UCT) and Artificial Neural Networks (ANNs):

##### Description: UCT is an algorithm used for decision making in AI, often combined with ANNs for better performance. UCT uses exploration and exploitation to make decisions, while ANNs learn and adapt through training.

##### Advantages: This combination allows the system to make robust decisions in complex situations and adapt over time as it gains more data.

##### Disadvantages: The implementation in Unity may require advanced understanding of both UCT and ANNs, and can be computationally intensive.

##### Self-Organizing System and Artificial Neural Networks:

##### Description: This combination uses neural networks for learning and adaptation, while a self-organizing system allows for the creation of complex emergent behavior.

##### Advantages: Can create highly complex and adaptive behaviors, and is capable of organizing large amounts of data.

##### Disadvantages: It can be complex to implement in Unity, and the emergent behavior may be hard to predict and control. Additionally, training and setup can be computationally demanding.

**General Conclusion**

Dynamic scripting will be used as a baseline test for DDA implementation due to it being feasible to implement and not as computationally expensive as other methods. If time allows, perhaps one other DDA method can be evaluated to be implemented.

**Implementation Mechanics**

Dynamically adjusting

* Number of enemies
* Generation of ammo, health and damage pickups
* Speed and Damage of enemies depending on time OR player performance
* Enemy behavior of dodging right and left

**High-Level Implementation Details**

* Fitness function
  + w1\*(Score/Kills) + w2\*(APM) + w3\*(Hit/Miss ratio) + w5\*(Max Health - Damage Taken)/Max Health + w6\*(1/Time Elapsed)
  + Game State Information: Needs to be aware of the current state of the game
    - Current enemy numbers
    - Position and health of player and enemies (\*\*)
    - Available pickups
    - Items picked up
    - Damage of players
    - Hit/Miss Ratio
* Rules Set (Dynamic Scripting)
  + **Number of Enemies**
    - If player's score per minute increases, increase number of enemies.
    - If player's health decreases significantly in a short span, decrease number of enemies.
  + **Generation of Ammo, Health, and Damage Pickups**
    - If player's ammo is low and misses are high, increase ammo pickups.
    - If player's health is below a certain threshold, increase health pickups.
    - If player's score per minute and kills are high, increase damage pickups.
  + **Speed of Enemies**
    - If player's hit/miss ratio is high, increase enemy speed.
    - If player's hit/miss ratio is low, decrease enemy speed.
  + **Enemy Behavior (Dodging Right and Left)**
    - If player's hit ratio is high, increase enemy dodge behavior.
    - If player's hit ratio is low, decrease enemy dodge behavior.
* Adjustment Mechanism - weight of a specific ruleset
  + New weight = old weight + learning rate \* (reward - old weight)
    - Learning rate - preselected constant between 0 and 1
    - Rewards is 1 if a player's performance continues to improve and 0 otherwise.

**References**

<https://www.hindawi.com/journals/ahci/2018/5681652/>

Journal of DDA approaches

Minor References

<https://natureofcode.com/book/chapter-9-the-evolution-of-code/>

Programming genetic algorithm

<https://forum.unity.com/threads/tutorial-genetic-algorithm-c.479062/>

Genetic Algorithm in Unity & Saving

<https://medium.com/analytics-vidhya/genetic-algorithm-in-unity-using-c-72f0fafb535c>

Simple Genetic Algorithm in Unity