Adaptive Enemy AI Using Q-Learning in Unreal Engine 5

By George Kotti – CSE 4633 – April 2025

Q-Learning Overview

- Model-free reinforcement learning
- Q-Table: stores values for (State, Action) pairs
- TD Update Rule used to improve over time
- ε-greedy action selection

$$\text{New } Q(s,a) = Q(s,a) + \alpha\,[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)]$$

- New Q Value for that state and the action
- Learning Rate
- Reward for taking that action at that state
- Current Q Values
- Maximum expected future reward given the new state (s') and all possible actions at that new state.
- Discount Rate

Project Setup in UE5

- Environment setup (starting level)
- Actor Assets and Animations
- Actor BP and ABP setup
- Character Inheritance System
- Full Combat System
- JSON save/load system for Q-Table persistence

Project Setup Files

Category	Assets / Classes
Breakables	BreakableActor
Characters	BaseCharacter, CharacterTypes, KnightCharacter, KnightAnimInstance, SlashCharacter, SlashAnimInstance
Components	AttributeComponent
Enemies	Enemy, AQLearningEnemy
HUD	HealthBar, HealthBarComponent
Interfaces	HitInterface
Items	Item, Treasure, Weapon

Category	Assets / Classes
Q-Learning	QLearningManager, QLearningTypes, QLearningEnemy
Referenced	BaseCharacter, KnightCharacter, Enemy, Weapon

Inside Editor:

- Animation Montages
- Animation Blueprints
- Animation Notify system
- Actor Blueprints
- Parameter tweaking
- Socket setup
- Level setup

State and Action Space

- State Features:
  - Health values, Distance action flags, Player action flags

- Actions:
  - Attack, Guard, Dodge, Heal, Wait

- Q-Table shape:
  - Tmap<FQState, TMap<EQAction, float>>

State features and actions are modular and decoupled from specific objects, allowing them to be easily extended or replaced.

Training and Exploration

- Rewards for good behavior: successful hits, dodges
- Penalties for bad behavior: attacking outside
- ε-greedy used to ensure exploration
- Updates every 1s asynchronously (or after notify event)

```
                 - Reward Values -

Implemented
Successful Attack: +1          (in Knight Get_Hit)
Successful Dodge: +0.25        (QEnemy OnDodgeEnd)
Being Hit: -1                  (QEnemy Get_Hit)
Target Dies: +10               (in Knight Get_Hit or Die)
Self Dies: -10                 (QEnemy Get_Hit or Die)
Healing: +0.25                 (QEnemy SetToHealing)
Attacked outside Range: -1.f   (QEnemy)
Attacked inside Range: +1.f    (QEnemy)
Swing and miss: -0.75          (Weapon)
Dodge Target Attack: +0.5      (QEnemy)
Guard Target Attack: +0.35     (Weapon)
```

Results and Behavior

- Initially attacks randomly or too often
- Over time, learns to guard and sometimes dodge
- Avoids attacking out of range
- Q-values reflect learned preferences

Persistent Storage

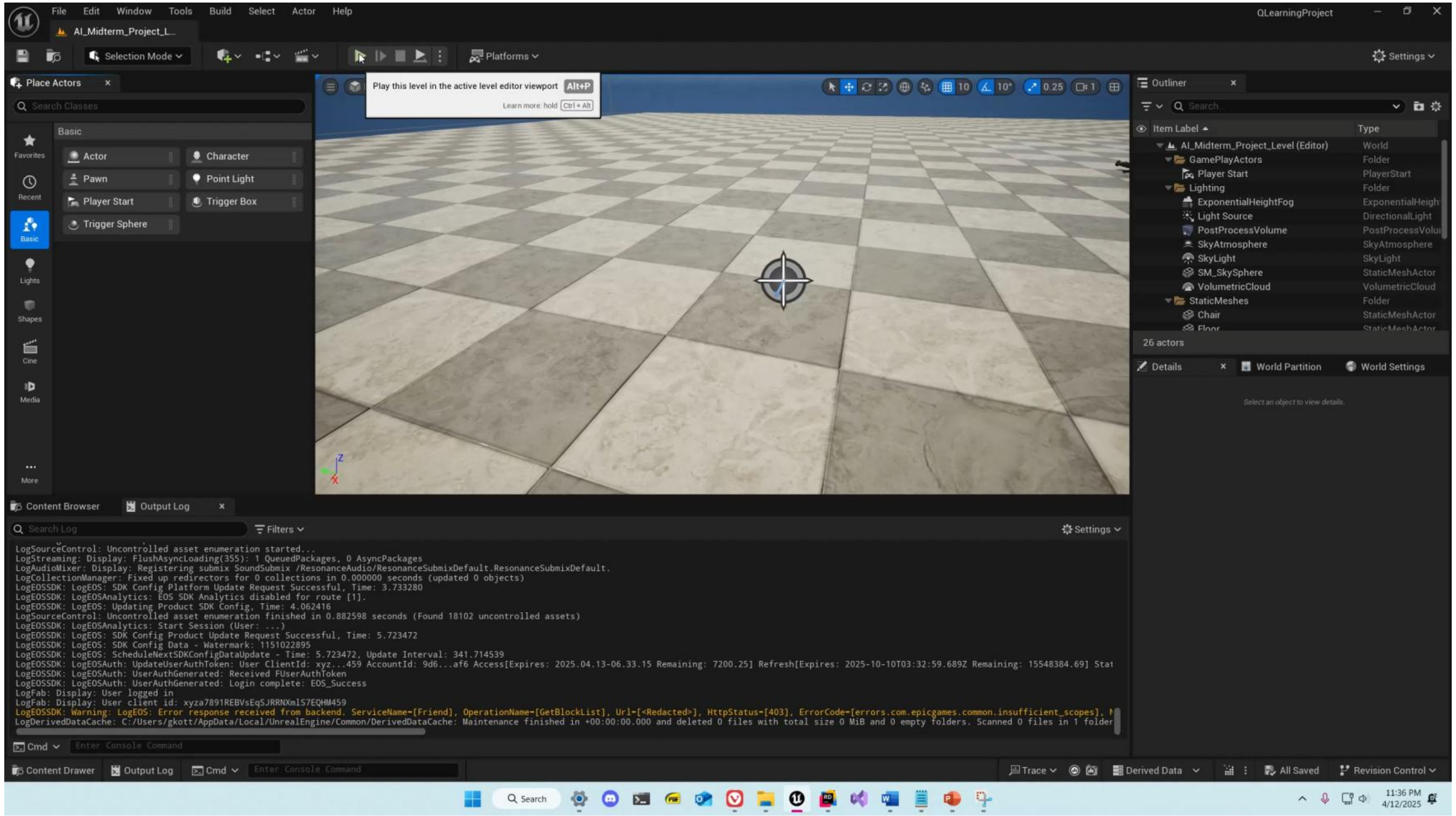
- Q-Table stored as a nested JSON structure
- Format: { StateString: { ActionEnum: QValue } }
- Allows saving and loading between sessions
- Supports shared learning via merged Q-tables (avg)
- Used FJsonObject, FFileHelper, and TJsonWriter in C++

What was learned

- Implementing RL in a real-time game involves asynchronous thinking and synching of events
- Action timing and animation blending affect Q-Learning updates.
- Game state design (discretization) impacts learning quality
- Smaller state/action spaces improve early training
- Rewards based on the timing of events is crucial for meaningful behavior
- Log extensively cause debugging null values is not where its at

Future Improvements or Expansions

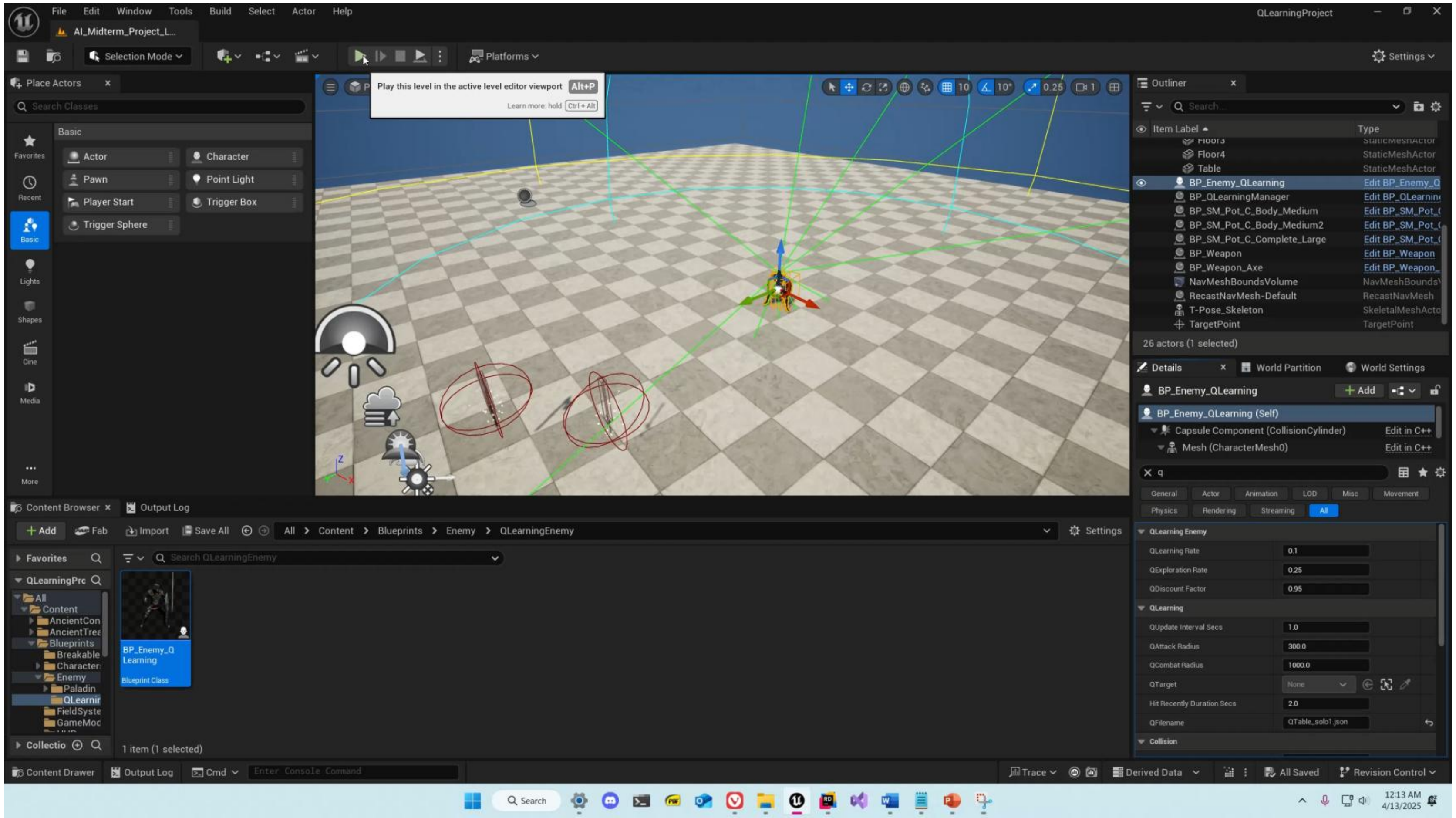
- Smarter reward shaping
- Deep Q-Network (DQN)
- Automated training
- Difficulty scaling based on player performance



Demo: Midway in Training

Demo: No Prior Training

Learns to Guard



Thanks!