Implementación y Análisis de Agente de Aprendizaje por Refuerzo en VizDoom: Deadly Corridor

Juan José Galindo Márquez

4 de diciembre de 2025

Resumen

Este documento detalla la teoría, detalles de implementación y resultados de un agente de Aprendizaje por Refuerzo Profundo (Deep RL) entrenado para navegar y combatir en el entorno VizDoom, específicamente en el escenario "Deadly Corridor". Se analiza la selección del algoritmo PPO frente a alternativas como DQN, la ingeniería de recompensas (Reward Shaping), la evolución de arquitecturas desde CNNs con apilamiento de fotogramas hasta políticas recurrentes (LSTM), y se presenta una evaluación comparativa del rendimiento bajo diferentes niveles de dificultad.

Índice

1 Selección del Algoritmo: PPO vs. DQN

Para este proyecto, se seleccionó el algoritmo **Proximal Policy Optimization (PPO)** en lugar de enfoques basados en valor como *Deep Q-Networks (DQN)*. La elección se fundamenta en las características intrínsecas del entorno Doom[2] (3D, parcialmente observable) y la naturaleza de las acciones requeridas.

1.1 Limitaciones de DQN en Entornos FPS

DQN es un algoritmo *off-policy* que aprende una función de valor $Q(s, a)$.[3] Aunque es eficiente en el uso de muestras, presenta desventajas críticas para este escenario:

- **Espacios de Acción Estocásticos:** En Doom, el movimiento óptimo a menudo requiere suavidad y cierto grado de aleatoriedad controlada (estocasticidad) para evitar ser un blanco fácil. DQN aprende políticas deterministas ($argmax_a Q(s, a)$), lo que resulta en movimientos robóticos y predecibles.
- **Estabilidad:** DQN es propenso a la inestabilidad en entornos con recompensas ruidosas y grandes variaciones visuales, requiriendo un ajuste fino de la tasa de exploración (ϵ-greedy).

1.2 Ventajas de PPO (Algoritmo Seleccionado)

PPO es un algoritmo *on-policy* que optimiza directamente la política $\pi_\theta(a|s)$. Lo preferimos por tres razones principales:

1. **Región de Confianza (Trust Region):** PPO introduce un mecanismo de 'clipping' en la función objetivo que evita actualizaciones destructivas de los pesos de la red. Esto es crucial cuando el agente recibe recompensas masivas (ej. +150 por un "Kill") que podrían desestabilizar el aprendizaje.
2. **Manejo de Políticas Continuas/Discretas:** PPO maneja naturalmente la distribución de probabilidades de las acciones, permitiendo al agente explorar combinaciones complejas (como *strafing* + disparo) de manera más orgánica.
3. **Integración con Recurrencia:** PPO se adapta mejor a arquitecturas recurrentes (RecurrentPPO) ya que aprende trayectorias completas, lo cual es vital para manejar la memoria en entornos parcialmente observables.

2 Estrategia de Ingeniería de Recompensas (Reward Shaping)

El entorno base de 'Deadly Corridor' otorga recompensas muy pobres (solo por acercarse al objetivo, el chaleco verde en este caso). Para acelerar la convergencia y fomentar comportamientos tácticos, implementamos un esquema de *Reward Shaping* denso:

2.1 Componentes de la Recompensa

- **Living Penalty (-0.01 por unidad de tiempo transcurrida):** Se aplica una pequeña penalización constante para incentivar al agente a alcanzar el objetivo rápidamente, mitigando el problema del 'agente perezoso' que prefiere quedarse quieto para no perder salud.

- **Incentivo de Combate (+35 a +150 por Kill):** Se otorga una recompensa positiva significativa al eliminar enemigos. Esto transforma al agente de un 'pacifista corredor' (que muere por la espalda) a un "limpiador táctico". La magnitud se ajustó durante el entrenamiento; valores muy altos (+150) provocaban comportamientos suicidas, mientras que valores moderados (+35) fomentan la supervivencia.
- **Penalización por Dolor ($2x$ de Daño Recibido):** Se multiplica el daño recibido por un factor (ej. 2.0). En RL estándar, perder 10 HP es un costo bajo comparado con ganar el episodio. Al amplificar este costo, forzamos matemáticamente al agente a valorar el *Strafing* (esquivar) sobre el "tanqueo"de daño.

2.2 Normalización (VecNormalize)

Dado que nuestras recompensas oscilan drásticamente (de -40 por dolor a +150 por muerte), aplicamos `VecNormalize`. Esto escala las recompensas a una media de 0 y varianza de 1, permitiendo que el optimizador (Adam) funcione correctamente sin sufrir de gradientes explosivos.

3 Arquitecturas de Redes Neuronales

Se proponen dos tipos de arquitecturas distintas para la extracción de información espacio-temporal del espacio observacional del ambiente:[1, 4, 6]

3.1 CNN con Apilamiento de Fotogramas (Frame Stacking)

Inicialmente, utilizamos una arquitectura basada en **CnnPolicy** con un apilamiento de 4 fotogramas (*Frame Stacking*).

- **Estructura:** La entrada a la red no es una imagen $(H, W, 3)$, sino un volumen de $(H, W, 12)$ (3 canales RGB $\times$ 4 fotogramas apilados).
- **Ventajas:** Permite a una red convolucional estándar percibir **detalles espaciales y cambios ligeros en el espacio**. Al ver 4 cuadros simultáneos, la red puede intuir en un corto período de tiempo, la noción de movimiento y secuencialidad.
- **Desventajas:** Carece de memoria a largo plazo. Si un enemigo se oculta detrás de un pilar por más de 4 cuadros, desaparece de la "mente"del agente, quien asume que la amenaza ya no existe. Además, aumenta la carga computacional de la primera capa convolucional.

3.2 Fase 2: RecurrentPPO (CNN + LSTM)

Para superar la limitación de la memoria a corto plazo, migramos a una arquitectura **Recurrente (CnnLstmPolicy)**.

- **Estructura:** Utilizamos un extractor de características *Custom ResNet* que alimenta una celda de memoria LSTM (Long Short-Term Memory) con 256 unidades ocultas.
- **Extractor Visual:** A diferencia de la *NatureCNN* estándar (3 capas), implementamos una red más profunda con *Batch Normalization* y *LeakyReLU*. Esto previene el problema de "neuronas muertas", común en imágenes de Doom con muchas zonas negras/oscuras.

- **Ventajas:** Proporciona una **noción mejorada de secuencialidad**. El vector de estado oculto h_t mantiene información sobre enemigos vistos en fotogramas pasados, permitiendo al agente anticipar emboscadas o recordar"la ubicación de enemigos ocultos. Esto elimina la necesidad de apilar 4 fotogramas, reduciendo la dimensión de entrada.
- **Desventajas:** Las LSTM son redes neuronales más críticas en el manejo de gradientes y además, tienen un aprendizaje lento en este ambiente. El uso de esta arquitectura, aumenta considerablemente el tiempo de entrenamiento.

4 Criterios de Entrenamiento y Función de Pérdida

El entrenamiento se rige por la minimización de una función de pérdida compuesta que equilibra la mejora de la política, la precisión de la estimación de valor y la exploración.

$$L_{total} = L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S[\pi_\theta](s) \tag{1}$$

[5]

1. **Pérdida de Política Recortada (L^{CLIP}):** Utilizamos un *clip range* (ϵ) de 0.2, esto limita cuánto puede cambiar la política en una sola actualización. En un episodio cualquiera, el agente podría tener suerte y obtener una recompensa masiva sin necesariamente haber cursado una estrategia óptima a largo plazo. Sin el recorte, la red aprendería drásticamente esta estrategia subóptima. El recorte asegura actualizaciones conservadoras y estables.
2. **Pérdida de Valor (L^{VF}):** Es el Error Cuadrático Medio (MSE) entre la predicción del Crítico y la recompensa real obtenida. Un Crítico preciso es vital para reducir la varianza. Por este motivo, usamos un **GAE (Generalized Advantage Estimation)** con $\lambda = 0{,}95$ para suavizar la recompensa a largo plazo, ayudando al agente a asociar acciones tempranas con recompensas tardías.
3. **Entropía (S):** Utilizamos un coeficiente de entropía (`ent_coef`) de **0.03**. VizDoom es propenso a óptimos locales (ej. quedarse quieto en la sala de inicio). La bonificación por entropía fuerza al agente a mantener cierta aleatoriedad en sus acciones, asegurando que siga intentando estrategias de exploración distintas, de manera que no se estanque en la explotación instantánea de estrategias subóptimas.

5 Análisis Comparativo de Rendimiento

Para validar la eficacia del modelo con estrategia PPO y arquitectura convolucional con apilamiento de 4 fotogramas, realizamos una evaluación comparativa contra un agente de política netamente aleatoria en tres niveles de dificultad del escenario.

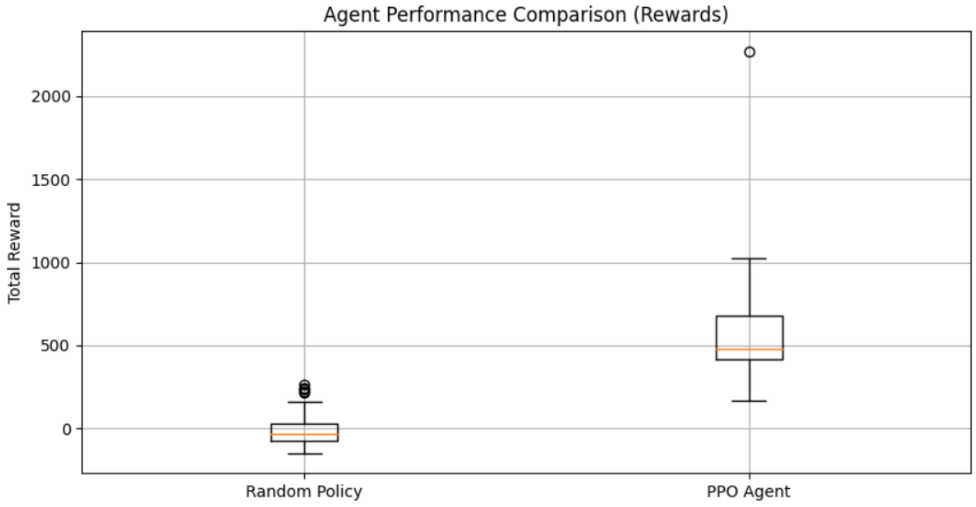
5.1 Metodología

Se ejecutaron 100 episodios para cada modelo en las dificultades 1 (I'm too young to die), 3 (Hurt me plenty) y 5 (Nightmare). Se midió la recompensa total acumulada por episodio , con lo que se extrajo la media y la desviación estándar asociada a cada estrategia por dificultad.

5.2 Resultados Gráficos (BoxPlots)


```
Random Agent: Mean=231.79, Standard Deviation=192.48
PPO Agent:    Mean=1212.50, Standard Deviation=686.43
```

Figura 1: Comparación de medias y desviación estándar - Dificultad 1




```
Random Agent: Mean=-13.42, Standard Deviation=89.27
PPO Agent:    Mean=558.70, Standard Deviation=248.26
```

Figura 2: Comparación de medias y desviación estándar - Dificultad 3

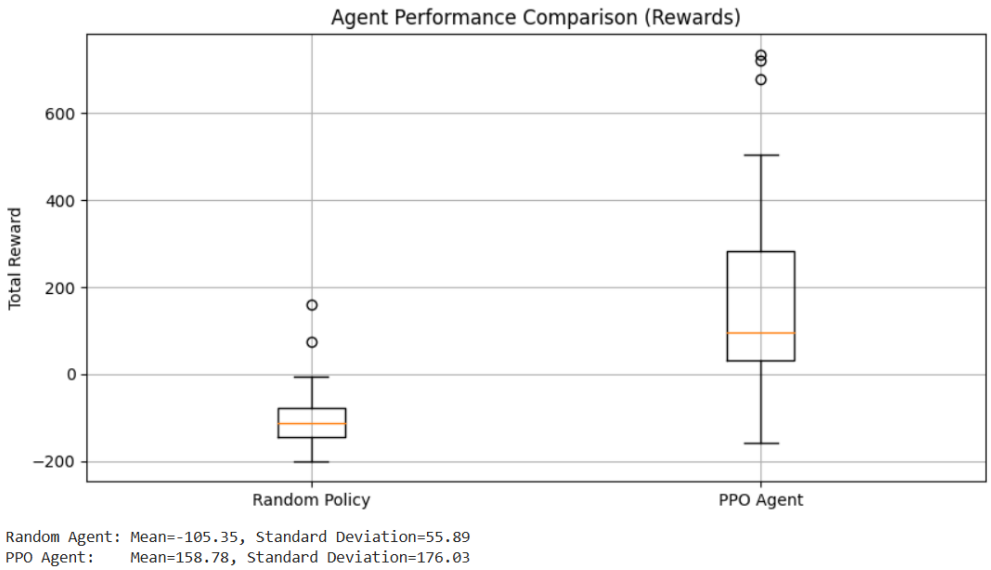


Figura 3: Comparación de medias y desviación estándar - Dificultad 5

5.3 Interpretación

El agente aleatorio muestra consistentemente un desempeño pobre ($\mu_{aleatorio} < \mu_{PPO}$), muriendo rápidamente o agotando el tiempo. El modelo PPO, aunque sufre de mayor desviación estándar (fenómeno de "todo o nada"), logra completar el objetivo en un porcentaje significativo de intentos, validando la arquitectura convolucional con apilamiento de fotogramas y la estrategia de *Reward Shaping* implementada.

Visualmente, se logra evidenciar que la estrategia de *Reward Shaping*, lleva de forma efectiva a que el agente busque vencer a los enemigos que tiene en frente; sin embargo, la configuración de esta recompensa parece ser bastante agresiva, con ello se explica la alta desviación estándar de la recompensa a lo largo de los 100 episodios; puesto que, en general, el agente es bastante agresivo; así, en algunos episodios el agente correrá con la suerte de matar múltiples enemigos y en otros no, por lo que se genera la alta dispersión en la recompensa.

Referencias

[1] Doomreinforcementlearning/vizdoom-deadlycorridor-tutorial. Ipynb at main · nicknochnack/doomreinforcementlearning. URL https://github.com/nicknochnack/DoomReinforcementLearning/blob/main/VizDoom-DeadlyCorridor-Tutorial.ipynb.

[2] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: a doom-based ai research platform for visual reinforcement learning. 2016. doi: 10.48550/ARXIV.1605.02097. URL https://arxiv.org/abs/1605.02097.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://www.nature.com/articles/nature14236.

[4] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. ISSN 1533-7928. URL http://jmlr.org/papers/v22/20-1364.html.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[6] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: a standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032.