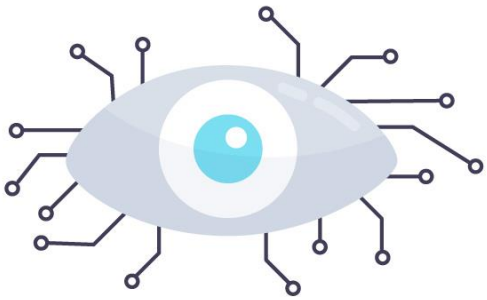
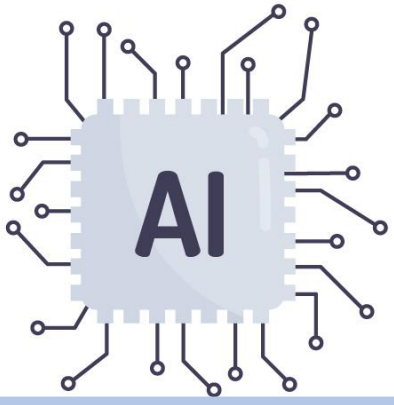
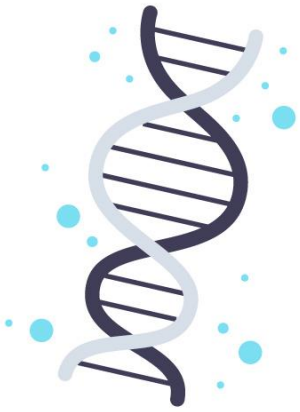






Proyecto P.U.M.A.

informe introducción a la ingeniería

Facultad De Ingeniería

Profesores:

- Ing. Ricardo Palma
- Lic. Patricia Stillger
- Ing. Pablo de Simone

Participantes:

- Jahaziel Ramos
- Josías Vidal
- Juan Moreno

Índice

Breve introducción al proyecto

La idea propuesta tiene como objetivo crear un autómata que pueda sondear, reaccionar, seguir ordenes, proveer información táctica para el combate y predecir comportamientos basados en un análisis estadístico combinado con aprendizaje de máquina para encajar los mismos en perfiles psicológicos aumentando que tan bien predice durante el cursado de su accionamiento.

El mismo está planteado en código como para aplicarlo en un videojuego, Counter Strike: Global Offensive, con intención de demostrar que es efectivo en ámbitos tanto de simulación como en la vida real.
El mismo tiene 4 particularidades que lo definen como tal:

- Percepción
- Ubicación
- Modus
- Atención

Este modelo y su desarrollo están orientados a formar parte del mismo como objetivo, crearlo de forma que se pueda integrar a cualquier nivel deseado como un módulo, fácil de implementar y modificar a gusto de un usuario y posiblemente creando un estándar nuevo en la industria para uso de redes neuronales en general.

Dicho esto, considérese que el modelo estará planteado en forma teórica con modelos genéricos de detección pre entrenados para muestreo y testeo de la idea como tal.

Reconocimiento de objetivos

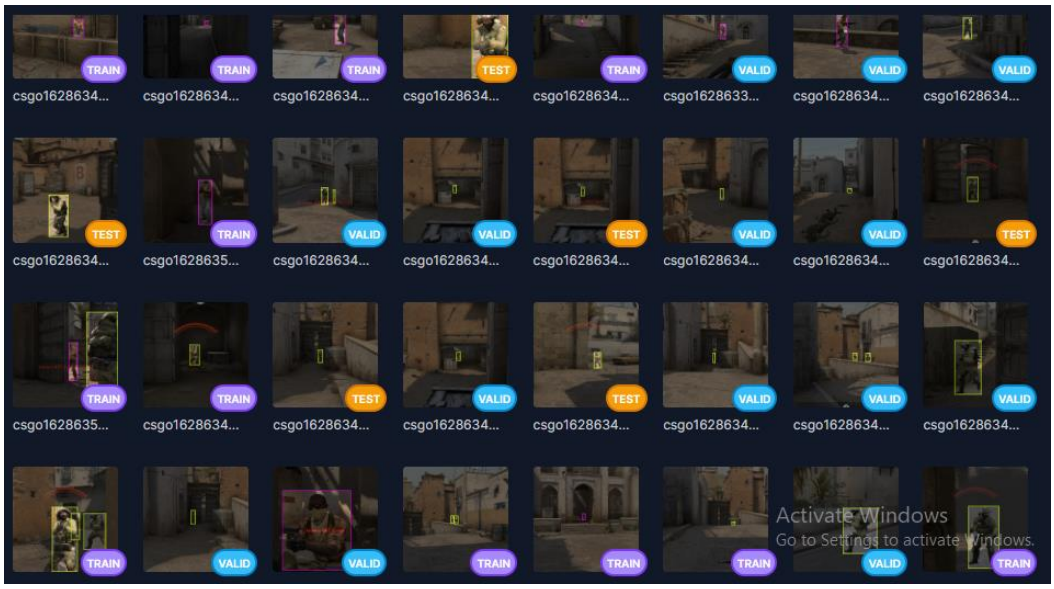
debe prepararse para su aplicación un set de imágenes etiquetadas adecuadamente para entrenar el modelo

ya que debe poder identificar aliados de enemigos, a distancias de poco renderizado, por ende el modelo idealmente debería ser entrenado con un data set de imágenes con etiquetado de aliados y enemigos.

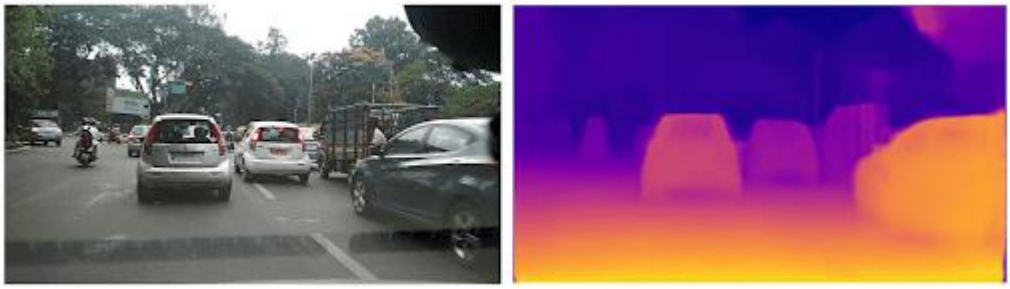
El modelo utilizado en el código fuente es yolov3 320x320, un modelo genérico algo anticuado utilizado para detección de varias clases de objetos, no es lo óptimo pero sirve como demostración de funcionamiento.

(https://pjreddie.com/darknet/yolo/)

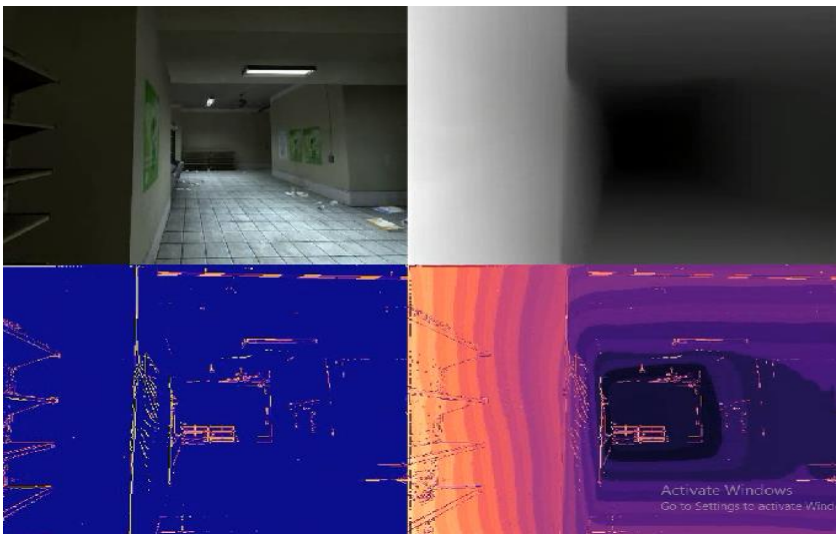
Lo ideal sería automatizar el proceso de toma de captura utilizando datos internos del juego o por lo menos un ambiente pre-calculado para simulación, con tal de tener fiabilidad a la hora de comparar los datos y que el modelo aprenda correctamente. Los modelos generalmente están dispuestos sin tener en cuenta (en el caso de nuestra aplicación) que no siempre son los mismos modelos 3d del mismo lado, para lo cual el mismo juego en este caso y sus desarrolladores implementaron una flecha por encima de aliados para evitar confusiones y seguir el rastro de compañeros. Esto además de servir para evitar fuego aliado, sirve para cuando se tenga más de un autómata operando, sea más fiel la ubicación de sí mismo y de sus aliados, por medio de comunicación y coordinación través de los modelos de percepción de profundidad (topología).



Reconocimiento de Terreno, Superficies y Relieves



En la imagen de muestra se observa aplicado el modelo de detección de profundidad “MiDaS”, (github.com/isl-org/MiDaS).
Este modelo está entrenado para interpretar profundidad a partir de una sola imagen. Sin embargo, junto con un filtro para percibir bordes y contornos (mostrado en la siguiente imagen) puede interpretar de forma más precisa el entorno 3D en el que se encuentra.



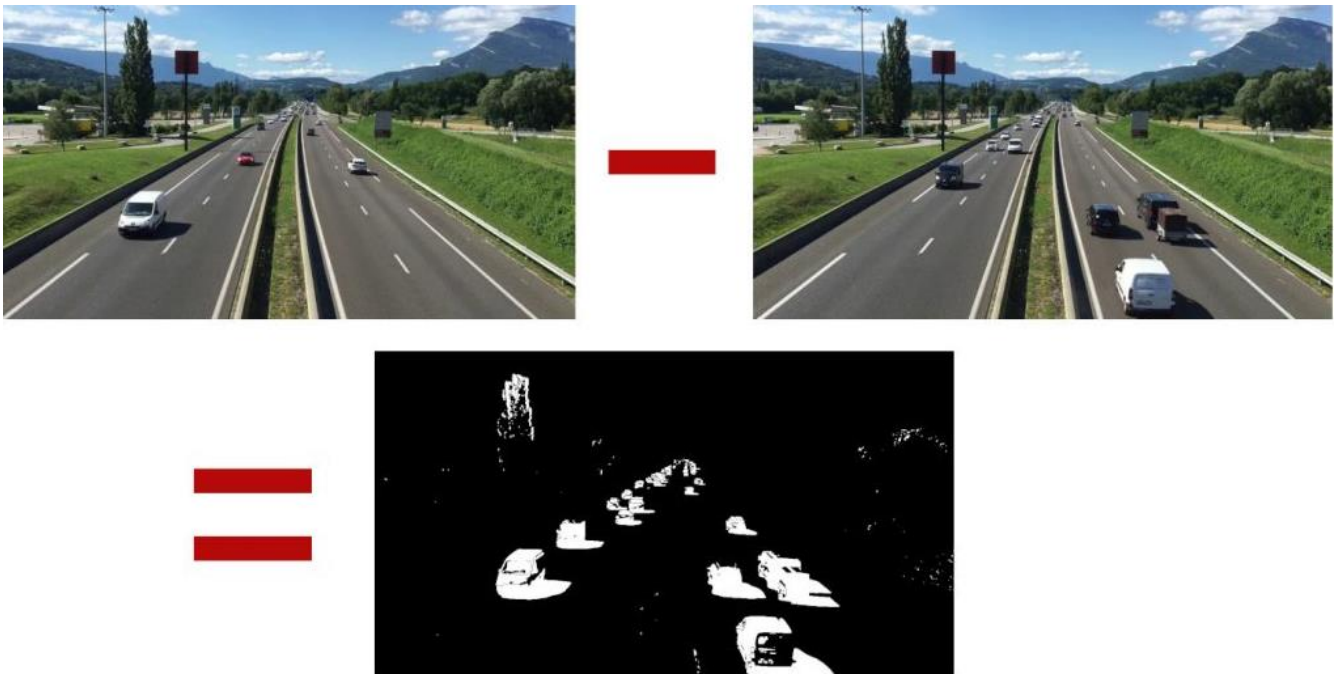
Aplicando el modelo a la imagen en vivo del autómata, podrá interpretar puntos de atención y ubicarse en vivo en su entorno, interpretando particularidades tal como son objetos (cajas, plantas, rejas) y relieves (ventanas, puertas),no solo como una referencia gráfica de la misma ubicación sino como posibles puntos de cobertura para sí mismo o nidos enemigos, y puntos de interés para lanzado de utilidades (granadas).

Esto será necesario con tal que el modelo sepa caminos de razonamiento y acción en el campo de batalla y tome decisiones en base a las posibilidades reales que tiene.

Reconocimiento de traslación de perspectiva

Se debe aplicar un filtro entre un fotograma y el siguiente para detectar movimientos absolutos (pixeles captados) respecto a movimientos relativos a perspectiva (el movimiento del autómata).

Esto servirá para detectar pequeñas irregularidades, interpretadas como enemigos asomando para atacar y/o utilidades aliadas/enemigas.



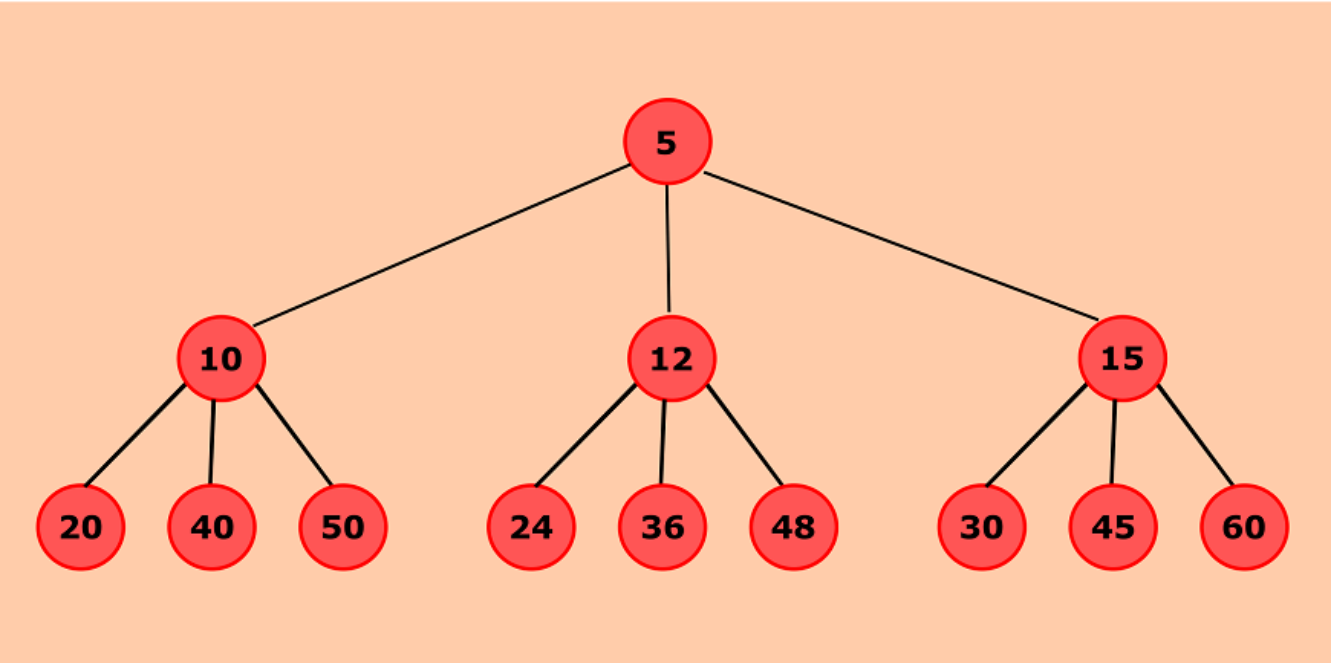
Se puede hacer comparación directa al usar la función opencv de absdiff() entre 2 capturas distintas, junto con un rastreo de contorno por trayectoria y diferencia en vivo de la posición calculada por el modelo para poder interpretar pixeles fuera de lugar, indicarían las irregularidades buscadas.

Interpretación de ordenes e interfaz de comando

Lo recomendable sería una interfaz activada por voz o macros del teclado, con diseño gráfico que se superponga sobre la pantalla del juego para facilitar la interpretación de la información que provee cada autómata a la hora de actuar.

Para interpretar órdenes, debe poder entender tanto comandos generales como mantener posición, cobertura para avanzar y tomar por la fuerza una posición X, así como comandos más específicos como lo son tirar utilidades, enfocar un ángulo en particular, etc.

Tal como está descrita la propuesta de razonamiento, se asemeja mucho a reordenar una lista de comportamientos a priori, como lo sería una “linked tree list”. Con cada elemento asignado dentro del orden de actividades una respuesta particular o punto de calor específico al cual prestarle atención según el interpretador en cuestión.

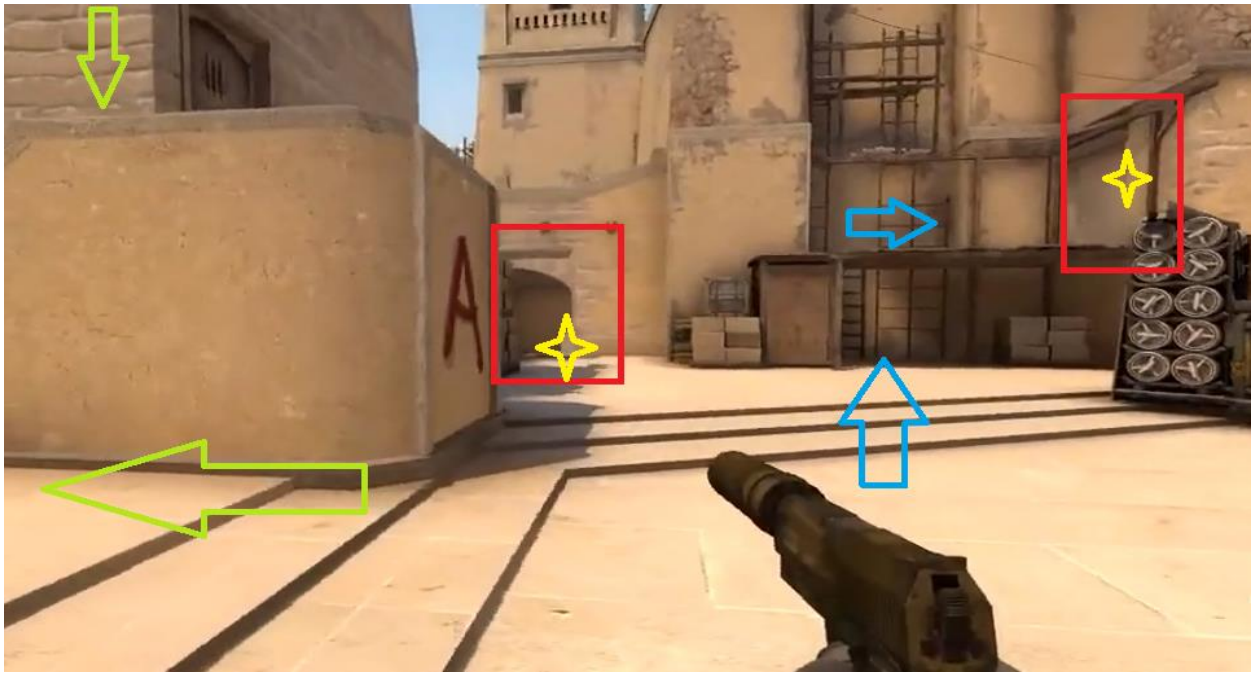


A la hora de interpretar cada orden, debe poder tener cierto nivel de independencia para decisiones sobre qué camino tomar, con un aprendizaje interno aplicado al modelo del árbol lógico interpretado por la máquina para entender el peso del nivel de prioridad de cada implicación de una orden que a su vez es estimado junto con un modelo interno de “game sense” que determina el mejor camino posible hacia poder tener éxito en su tarea

Decisión, preparación y aplicación de micro tácticas

En el transcurso de la partida, se presentan alternativas para proceder con el objetivo de ganar. Como se presentó anteriormente un árbol de razonamiento lógico, se debe plantear dependiendo de que situaciones se presentan qué camino tomar en cuanto a las acciones posibles de la posición actual.

Tómese como ejemplo la siguiente imagen:



las flechas verdes y azules indican cada color un camino alternativo, decidiendo que posición tomar respecto a que posición vigilar (cuadros en rojo con posible punto de irregularidad), en cuanto a jugadores es muy difícil coordinar quién es el que se dedica a mirar cada ángulo en cuestión de plazos cortos de tiempo. Es necesario al enfrentar esta situación táctica poder determinar lo más inmediato posible entre pares a la hora de rotar y cambiar ángulos de vigilancia con tal de poder reaccionar de la forma más inmediata posible a un potencial ataque enemigo.

Sin embargo no siempre se defiende o ataca, debe poder interpretarse también en base a que objetivo se plantea dicha situación, como lo es en este caso defender. Estas decisiones cambian según variados factores como lo son el mismo aprendizaje categorizado de perfilado del enemigo, utilidades enemigas bloqueando la vista o camino, órdenes y jugada óptima en cuanto a las mismas, entre otros.

Se concluye que el modelo de aprendizaje debe tomar en cuenta variados factores y retroalimentarse de ellos con tal de lograr una victoria, y a su vez poder coordinar con el operador y demás autómatas a la hora de tomar decisiones y realizar acciones/reacciones respecto a la situación del campo de batalla

Interpretación de variación en comportamientos ajenos

En cuanto respecta a los enemigos, si se analiza de cerca las tendencias por jugador y equipo, se puede perfilar cada jugador según su rendimiento y comportamiento. Esto servirá para determinar de forma más precisa las predicciones y así poder tener mejor probabilidad de éxito en micro tácticas.

De por sí, un análisis del rendimiento en jugadores y una forma fiel de comparar datos entre repeticiones de partidas serviría para construir un “esqueleto” o modelo en blanco básico con tal de verificar el correcto funcionamiento del autómata y a su vez comparar casos opuestos y perfecciona el modelo contra distintas situaciones con tal de estar calibrado a nivel competitivo.

Por ende, una buena manera de comparar estos resultados es analizar con un modelo con weights vacíos e ir entrenando el mismo con resultados grabados en partidas competitivas ya armadas, utilizándolas como base de datos bastante fiel para construir el armazón del modelo, preferiblemente renderizándolas en el mismo juego, capturando el video de cada jugador, asignando un valor fijo de rendimiento de ante mano según los resultados finales de la partida. Así, se tendrá una referencia básica fiel pero no aleatoria con tal de ahorrar tiempo en el entrenamiento.

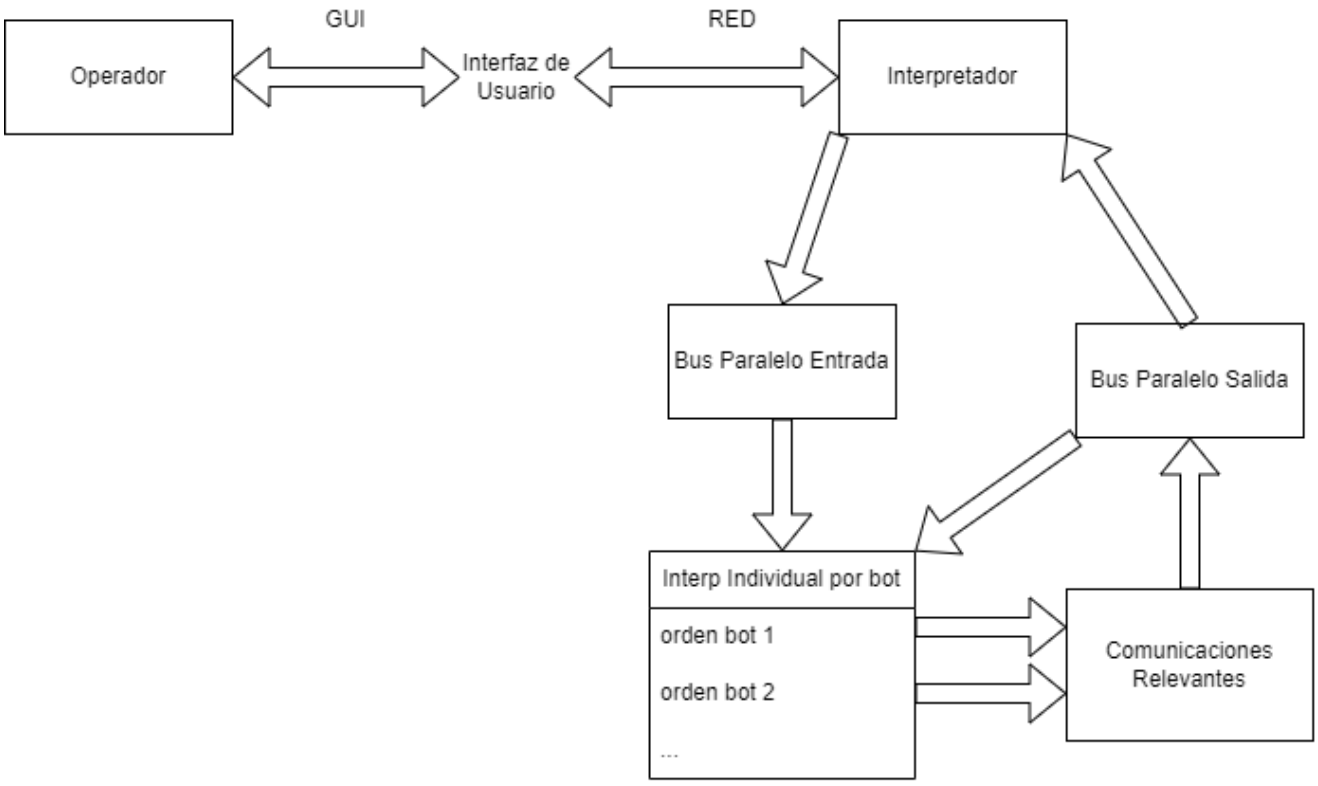
Otra forma sería un rastreo en vivo utilizando un juego modificado o “modeado” o incluso usando hooks del sistema operativo con tal de tener acceso directo por código a la información del juego y trabajar puramente con python a la hora de implementar las variables de la misma.

Comunicación operador/máquina y máquina/máquina

En cuanto al código para poder comunicarse, es preferible que se arme una API, con las llaves para acceder a la misma siendo generadas en el momento y los comandos siendo enviados a través de protocolo HTTPS por internet (en caso de ser remoto) o red privada.
respecto a la comunicación interna, lo ideal sería un bus de comunicación virtual análogo al protocolo I2C, donde se asignaría de forma síncrona la identificación de cada instancia del modelo, con tal de que el interpretador de orden sepa identificar a que autómata se refiere a la hora de dictar las mismas.

Esto permitirá asignar roles de posicionamiento y armamento, sincronizar ataques/defensas y mayor percepción del campo al facilitar y acelerar notablemente el ritmo al cual se entiende la situación táctica, pudiendo así reaccionar de forma inmediata a cualquier cambio.

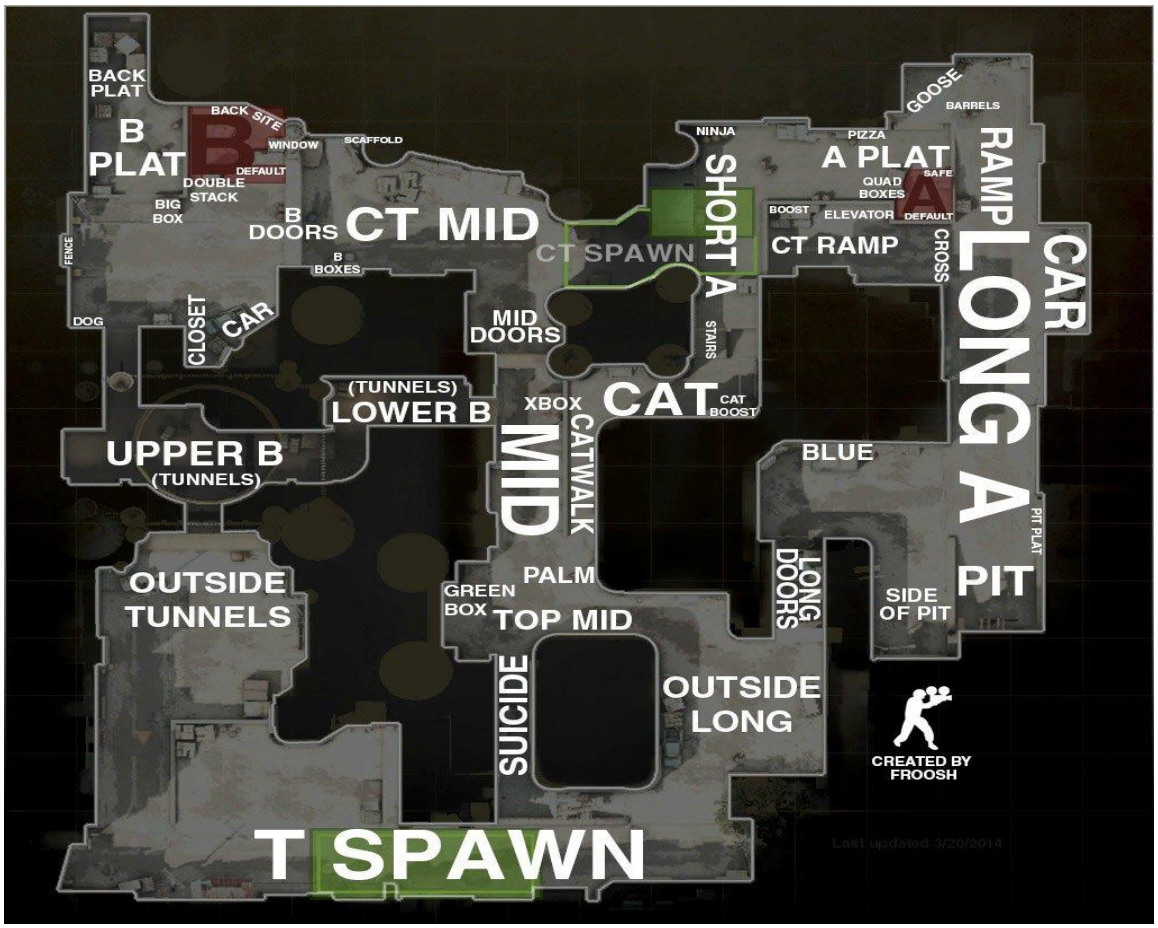
Se muestra un aproximado esquema de comunicación en la siguiente imagen:



Trazado de trayectoria y accionamiento aproximado según predicciones

Parte del entendimiento que debe ser provisto al modelo para poder no solo comportarse de acuerdo a sus posibilidades reales interpretadas por la imagen sino también para poder entender y memorizar patrones propios y del enemigo respecto al movimiento del mapa, es el trazado de los mismos.

Ya provisto el modelo de percepción de profundidad, más la detección de relieves y superficies a través de filtros, se pueden hacer mapas de relaciones respecto a posiciones que son clave en el mapa para poder lograr los objetivos de cada equipo.



esto permitirá tener un mapeo en vivo a la hora de que el modelo aprenda de las experiencias de las partidas tanto realizadas como simuladas por repeticiones de partidas profesionales

Trazado de trayectoria de utilidades

Al considerar el hecho de que las utilidades en situaciones tácticas, sean en un juego cualquiera o la vida real, llevan a una ventaja directa a la hora de ejecutar micro tácticas, no es de sorprenderse que se encuentren embebidas en las acciones de operadores con tal de poder entrar a un sitio objetivo, despejar el mismo de enemigos o usarlas como cobertura para confundirlos. Por lo tanto, se implementará un cálculo de trayectoria simple pero efectivo con el sondeo 3d de imagen del mapa.
Funcionará de la siguiente manera:

- Se calcula primero la velocidad inicial del personaje y su dirección.
- Se aproxima la velocidad de lanzamiento que depende del tipo de utilidad y su dirección según adonde se esté apuntando.
- De suman vectorialmente los vectores que describen la velocidad de ambos para generar un vector trayectoria inicial.
- Se calcula la desaceleración de la misma por gravedad del juego estimada.
- Se toman en cuenta los rebotes de la utilidad con un factor de resta de velocidad según el tipo de utilidad.
- Se utiliza un temporizador según el tipo de utilidad.
- Finalmente, entrega un resultado aproximado de punto de aplicación de la utilidad.

El vector amarillo representa la velocidad instantánea de la granada según el apuntado, el vector verde representa el movimiento del personaje instantáneo la trayectoria roja es la trayectoria de la granada, el vector azul es la constante de gravedad, la trayectoria morada es calculada a partir de un factor de rebote, finalmente la estrella blanca representa el estallido de la utilidad

Entrenamiento del sentido con estadística, modelo perfilador

Respecto al enemigo confrontado, se puede aplicar un aprendizaje profundo con tal de poder interpretar y reconocer mejor los perfiles de comportamiento de cada jugador teniendo en cuenta sus estadísticas y acciones durante el cursado de la partida, con tal de encajarlo en una base de datos ya pre-ensamblada.

En esto se debería consultar algún jugador profesional, con tal de poder incorporar la mayor cantidad posible de perfiles y poder acomodarlos en sub conjuntos con el propósito de poder perfilar durante el transcurso de la partida de forma más precisa cada enemigo.

Tómese un ejemplo hipotético, en el cual un jugador demuestra un comportamiento agresivo y acelerado respecto a una parte del mapa 3 rondas seguidas, utilizando armas de asalto especializadas en “rush”(sub metralletas, escopetas y micro tácticas de cegado/ confusión a distancias cercanas). Se plantea en el modelo de aprendizaje que este jugador respecto a esta área del mapa está tomando un rol “rusher” o de asalto. Caso opuesto respecto a la manera de enfrentar el enemigo sería un “lurker” o jugador sigiloso que se enfoca en ser impredecible a la hora de intentar cumplir un objetivo revelando la menor información posible, incluso llegando al extremo de tirar utilidades de amague con la intención de confundir su enemigo. Este perfil demuestra ser más paciente a la hora de tomar decisiones.

Estos roles serían la última categoría por la cual clasificar un jugador y poder predecir sus movimientos. Las categorías superiores se subdividirían respectivamente y acabarían en estos mismos roles para predecir comportamientos.

Es necesario consultar con gente dedicada al ámbito competitivo del juego para una correcta y completa categorización de todo el espectro de tipos de jugadores que hay.

Capacidad de enfocar zonas latentes y mejorar la imagen + Temporizador

Con la información provista del modelo de perfilado, junto a indicaciones de sonido provistas por el juego, se puede hacer énfasis en zonas particulares, las cuales se observan a distancia determinada por el modelado del mapa según fue percibido por imagen y el movimiento.
Tomando en cuenta comportamientos según predicciones del modelo, la situación siguiente cambia:

Según los datos provistos, hay 7 distintas posibles combinaciones de círculos rojos por los cuales el enemigo podría atacar en esta situación. Si está despejada la zona anterior al círculo de la izquierda, por ejemplo, es muy posible que se ataque por el círculo del medio y de la derecha, dependiendo del posicionamiento de los autómatas y el operador, se proveerá información que ayudará a determinar micro tácticas que retroalimentarán al modelo. Siendo el objetivo de esta programación que se refresque constantemente la información provista para que el modelo actúe de forma correcta e instantánea.
Sin embargo, no se toma en cuenta lo siguiente: EL TIEMPO, no es lo mismo esperar varios segundos en una partida que dura un minuto y medio máximo, con pocos autómatas restantes del equipo, a los ya pasados 50 segundos y no hay señales del otro sitio, a que sea la misma situación pero se tenga certeza que un jugador tomó el rol de lurker o sigiloso. O incluso la situación donde apenas pasaron 20 segundos. El tiempo determina también cuales posibilidades hay de ganar o por lo menos ahorrar recursos del juego, ya que hay un límite de tiempo. El temporizador interno es tan simple como conectar una variable de conteo directamente a una entrada particular de la red neuronal como si fuese un cronómetro. Que determinaría en el caso de defender una condición de victoria, o en el de atacar, una condición de derrota.

Implicaciones Éticas en otras aplicaciones

Posiblemente se argumente que un avance de esta naturaleza implique directamente una aplicación bélica y por ende seria cuestionable implementar o incluso desarrollar un proyecto de esta Naturaleza, Sin embargo, se puede argumentar todo lo contrario, ya que agregaría un aspecto tan refinado a la logística de la guerra, que probablemente termine eliminando aspectos de error humano y crueldad que de otra manera serían casi imposibles controlar.
Ejemplos como lo son PMCs en países inestables o intervenciones en los mismos por parte de la OTAN, serían menos crueles, menos sanguinarios y resultarían en una resolución de conflicto con la menor cantidad de casualidades y fuego colateral posible. Siendo este el mejor escenario al que podemos aspirar ya que, la guerra e incluso la violencia como tal, sigue presente en variados aspectos de nuestra realidad como sociedad. Un escenario donde realmente, la violencia tenga una naturaleza marcial y no cruel.

Mejoras en Hardware y Software

El rendimiento objetivo de 200 fotogramas por segundo requiere una implementación casi completa paralelizada y procesada por unidades de procesamiento gráfico (GPUs)

siendo la opción más prevalente, utilizar funciones de python que usan CUDA a la hora de implementar código, ya que aceleraría los programas de ejemplo provistos de ejecutarse en tiempos de orden de los segundos, a ordenes de microsegundos.
En cuanto el hardware, un estudio de rentabilidad para comparar rentar un cloud cluster (unidad en la nube) versus adquirir un equipo propio, debería realizarse para poder elegir de acuerdo a las posibilidades de comercializar.

Comercialización

En cuanto al modelo planteado y este proyecto en particular, se piensa en un servicio de entrenamiento para jugadores profesionales con tal de poder tener mayor tiempo disponible ya que no se necesitará esperar a coordinar partidas con equipos o entrenamientos, ya que se dispondrá de esta herramienta con tan solo una aplicación de escritorio que incluso sus aspectos gráficos y estadísticos puedan ser personalizados, ya que solo se requeriría una llave API por la cual se deba pagar una cuota mensual.

Particularidades y dificultades del desarrollo

Ya que no hay un camino pre ensamblado para algo tan complejo y específico como lo que se plantea en este proyecto, se debe armar una especie de monstruo de Frankenstein a la hora de escribir código funcional de ejemplo. Por una parte, es muy didáctico, pero por otra es frustrante ya que uno se topa con problemas de compatibilidad, errores y demás cuestiones, solo por una parte del código ya que hay mucha documentación sin actualizar (ni hablar de drivers, AMD no actualiza nada de ROCM hace más de 4 años y las gráficas NVIDIA están inalcanzables para un estudiante en cuanto lo que es precio). Eso junto a fechas variadas de lo que presentan los algoritmos de búsqueda hace que el desarrollo sea lento, siendo el aspecto rescatable de ello que uno aprende las particularidades de trabajar con redes neuronales que sean rápidas a la hora de realizar una tarea.