ISSN 1870-4069

Implementación de agentes mediante Máquinas de Estado Finito de Comportamiento (MEFC) aplicados a la simulación de multitudes

José Alberto Hernández Aguilar[1], Isaac Rudomín Goldberg[2], José Crispín Zavala Díaz[1], Leonel Toledo[2]

[1] Universidad Autónoma del Estado de Morelos, FCAeI, Cuernavaca, Morelos, México
[2] Barcelona Super Computer Center, Barcelona, España
jose_hernandez@uaem.mx

Resumen. En esta investigación se discute la simulación de multitudes y la generación de agentes mediante máquinas de estado finito (FSM) en XML, para este propósito se revisan herramientas open source líderes en esta área, particularmente se analizan Menge y FlameGPU, para cada una de ellas se realiza un análisis detallado del funcionamiento de XML aplicado para la generación de scripts de simulación. Luego del estudio comparativo, se propone un nuevo esquema general de XML para la definición de escenas y Máquinas de Estado Finito de Comportamiento denominado XBFSM (eXtendend Behavior Finite State Machine) que busca normalizar la forma en cómo operan cada una de estas plataformas. Nuestros resultados preliminares indican que la simulación de multitudes puede ser representada mediante una jerarquía de archivos XML, constituida por al menos dos esquemas XML, uno que describe el escenario y otro que establece el funcionamiento de la Máquina de Estado Finito de Comportamiento. Para un número pequeño de agentes se recomienda omitir los esquemas XML para la configurar las GPUs, mientras que para un número muy grande de agentes se recomienda configurar los esquemas XML que activan las GPUs.

Palabras clave: simulación de multitudes, XML, máquinas de estado finito, comportamiento, agentes inteligentes.

Implementation of Agents through Behavior Finite State Machines (BFSM) Applied to Crowd Simulation

Abstract. In this research we discuss the simulation of crowds and the generation of agents using finite state machines (FSM) in XML, for this purpose we review leading open source tools in this area, particularly Menge and FlameGPU are analyzed, for each of them a detailed analysis of the functioning of XML applied for the generation of simulation scripts is performed. After the comparative study, a new general scheme of XML is

proposed for the definition of scenes and Finite Behavior State Machines called XBFSM (eXtended Behavior Finite State Machine) that seeks to normalize the way in which each of these platforms operate. Our preliminary results indicate that crowd simulation can be represented by an XML file hierarchy, consisting of at least two XML schemas, one that describes the scenario and another that establishes the operation of the Behavior Finite State Machine. For a small number of agents it is recommended to omit the XML schemas for configuring the GPUs, while for a very large number of agents it is recommended to configure the XML schemas that activate the GPUs.

Keywords: crowd simulation, XML, finite state machines, behavior, intelligent agents.

1. Introducción

La Simulación de multitudes (Crowd Simulation en Inglés), más que una técnica de modelado, es un arte [5], algunas de las aplicaciones de la simulación de multitudes incluyen la detección de comportamiento de la población en lugares públicos como en un aeropuerto, una estación de trenes, o un partido de fútbol [4], su aplicación en la industria del cine se ha enfocado a la creación de grandes batallas (p.e. 300, Game of Thrones), invasiones humanas (por ejemplo: Guerra Mundial Z) y extraterrestres (p.e.: Día de la independencia). De acuerdo a este mismo autor "El objetivo principal en el estudio y simulación de multitudes es representar con precisión grupos de personajes autónomos, llamados agentes virtuales, con reglas y entornos lo más cercanos a aquellos encontrados en la vida real" [4].

En este sentido en [9] se discute un modelo para la simulación del flujo de caminantes y multitudes, este modelo se basa en fuerzas tales como: el deseo de alcanzar un lugar en un determinado tiempo, fuerzas para evitar la colisión entre caminantes, fuerzas para evitar paredes y obstáculos, donde a excepción de la fuerza del deseo, las situaciones se ven afectadas por el resultado del algoritmo del vecino más cercano.

El problema del vecino más cercano se puede resolver mediante la triangulación de Delaunay de manera incremental, en la cual se actualiza al paso del tiempo la posición de cada caminante y la *grid* de la que forma parte, sus resultados muestran que el modelo se desempeña bien para dinámicas como filas, adelantamiento, evitar obstáculos y comportamiento de pánico. En la tabla 1 se muestran los desarrollos más recientes en *Crowd simulation* y agentes.

1.1. Problema en manos

El problema en manos se ha formulado a partir de la siguiente pregunta de investigación: ¿Cómo se puede aplicar una BFSM (*Behavior Finite State Machine* – Máquina de Estados Finitos de Comportamiento) para la creación de agentes inteligentes y la simulación de multitudes en diferentes contextos?

Tabla 1. Desarrollos más recientes en Simulación de multitudes usando Agentes.

Autores	Descripción	Técnica
Luo, Chai, Ma, Zhou y Cai (2018)	Presentan un framework (marco de trabajo) para modelado basado en el comportamiento para modelar el proceso de cómo los humanos seleccionan y ejecutan proactivamente una estrategia de dirección en situaciones de mucha gente y evalúan el correspondiente comportamiento [10].	Modelado de Estrategias proactivas de dirección (proactive crowd) para minimizar colisiones potenciales basadas en comportamiento (gap seeking and following)
Karbovskii, Voloshin, Karsakov, Bezgodov & Gershenson (2018)	Discuten una técnica de simulación multi modelo basada en agentes que incorpora múltiples módulos. Esta integración se basa en un espacio abstracto común donde las entidades de los diferentes modelos interactúan y los agentes pueden ser manejados por diferentes modelos [6].	Simulación multi modelo basada en agentes. Se utiliza la presión de la multitud para estimar la capacidad de diferentes condiciones emergentes que afectan traumáticamente a los caminantes en la multitud.
Liu, Liu, Zhang et al.,(2018)	Presentan una simulación de evacuación de una multitud que está basada en el conocimiento de la navegación y un mecanismo de control de dos etapas. En este trabajo se utiliza un framework de un algoritmo cultural multi población, el mecanismo de control está dividido en dos partes: el espacio de creencias y el espacio de la población. El espacio de la población en grupos o (sub poblaciones), y un líder es seleccionado para cada grupo de acuerdo a un valor de aptitud (fitness value). El espacio de creencias comprende múltiples agentes y una base de conocimiento [8].	Un framework de Algoritmos culturales donde cada agente corresponde a un grupo y obtiene la posición del líder a través de una función de aceptación y más tarde pasa la información a la base de conocimiento.

1.2. Estructura del documento

En la primera sección se discute brevemente la simulación de multitudes, el funcionamiento de XML y su aplicación para la simulación de multitudes, se hace énfasis en la especificación de una BFSM (Behavior Finite State Machine - Máquinas de estado finito de comportamiento), en la segunda sección se describe el funcionamiento de los agentes bajo el contexto de los software de simulación Menge

y FlameGPU, la tercera sección presenta la metodología y nuestro modelo propuesto, la cuarta sección presenta los resultados y discusión. Finalmente, se presentan las conclusiones, trabajos futuros y referencias.

2. XML y las máquinas de estado finito de comportamiento (BFSM)

Las multitudes humanas son un problema de discusión actual, su modelado y simulación ha sido utilizado como herramienta importante para analizar el impacto del comportamiento de masas en un amplio rango de aplicaciones incluyendo la planeación de la seguridad, el diseño de arquitectura, realidad virtual, sistemas de entrenamiento y militares, entre otros [10].

Un documento XML consiste de una serie de caracteres, algunos de los cuales forman caracteres de datos y algunos de los cuales forman marcas. Estas marcas codifican una descripción del plan y la estructura del documento, e incluye comentarios, etiquetas o delimitadores (p.e., etiquetas de inicio, de final y espacios en blanco), declaraciones (p.e., declaraciones del tipo de documento, declaraciones XML, declaraciones de texto) e instrucciones de procesamiento. Los caracteres de los datos comprenden el texto que no está etiquetado.

Una aplicación de software que consume los datos del documento debe ser capaz de examinar el documento, accede a su estructura y contenido (p.e., separar los datos de las etiquetas) y colocar los datos en una forma interna útil. El módulo de software que analiza un documento XML es generalmente llamado procesador XML y trabaja en beneficio de la aplicación [15].

En [10] se señala que el modelado de multitudes basado en agentes ha emergido como el más popular y poderoso debido a su capacidad para modelar comportamientos individuales heterogéneos, así mencionan que la dinámica de la multitud está determinada por el comportamiento de los agentes individuales y su interacción. En este mismo sentido en [14] se generan máquinas de estado finitas a partir de archivos XML que se guardan en imágenes para ser consultadas por los agentes. De acuerdo a [11] un autómata finito (AF) o máquina de Estados Finitos (FSM - Finite State Machine por sus siglas en inglés) es un modelo matemático de un sistema compuesto por: estados, transiciones y acciones.

Un estado almacena información del pasado. Una transición indica un cambio de estado y este es determinado por la condición que es necesaria cumplir para activar la transición. Una acción es una descripción de una actividad que es realizada en un momento dado. Las máquinas de estado finitas, junto con los algoritmos A* para encontrar rutas, y los algoritmos para evitar colisiones son consideradas como técnicas de Inteligencia Artificial aplicadas para la simulación de multitudes.

En [16] se combinaron mapas y scripts XML, los mapas se definen en el script XML usando la etiqueta <map>. Diversos tipos de mapas pueden ser definidos, por ejemplo: mapas estáticos: mapas de ruta, mapas de altura, mapas de textura y mapas de etiquetas pueden ser cargados por el usuario, o actualizado interactivamente. Otros mapas son dinámicos (colisión, visibilidad, nivel de detalle) y pueden ser utilizados

por el sistema si se desea. La combinación de FSM y scripts XML resultaron ser muy productivas.

En [16] se desarrolló la idea de incluir probabilidad, jerarquía y capas al sistema de simulación. Los esquemas probabilísticos incrementaron la diversidad aparente de comportamientos, como comportamiento para los caracteres individuales no está únicamente guiado por scripts rígidos, pero aun así modificados en forma inesperada, pero aun controlada, al seleccionar aleatoriamente ciertas transiciones dentro de la FSM. En [17] se discutió el comportamiento de las multitudes "Crowd Behavior" a partir del uso de XML e imágenes, las multitudes pueden construirse tomando en cuenta los fluidos como base.

Algunos sensores originalmente desarrollados para los fluidos se desarrollaron y reformularon para diferentes usos en la pieza. Sin embargo, el principal propósito de la nueva pieza es permitir al artista construir escenas pobladas por diferentes caracteres. La tecnología que fue usada para implementar las multitudes fue XML.

A continuación, se discuten los agentes estacionarios derivados de las FSM con comportamiento.

2.1. Agentes estacionarios

De acuerdo con [12] cualquier sistema que esta principalmente enfocado al movimiento de agentes debería considerar todos los agentes estacionarios para ser equivalente. El comportamiento de cada agente puede ser diferente pero también tener trayectorias idénticas por medio de su BFSM (Behavior Finite State Machine-Maquina de estados finitos de comportamiento). Dos agentes estacionarios podrían ocupar diferentes estados en la BFSM, representando diferentes actividades o condiciones mentales. La técnica de modelado basada en agentes es una técnica para la simulación computacional de sistemas complejos que interactúan a través de la especificación del comportamiento de un número de individuos autónomos que actúan simultáneamente [7]. Este enfoque en los individuos es más costoso desde el punto de vista computacional, pero provee una manera más natural y flexible para estudiar los ambientes que demuestran el comportamiento emergente.

3. Funcionamiento de XML bajo Menge y FlameGPU

3.1. Menge

Es un ambiente de trabajo con una plataforma extensible y modular que permite simular el movimiento de los agentes en una multitud o aglomeración. La arquitectura de Menge está inspirada en la descomposición del problema de simulación de multitudes en sub problemas o componentes (véase la Figura 1).

3.2. Elementos de la arquitectura

Menge es una arquitectura modular basada en el concepto de elementos. Un tipo de elemento define un aspecto particular de un sub problema. El tipo de elemento define una interface que puede utilizarse para proveer una solución particular. Cada tipo de

elemento puede tener un conjunto diverso de implementaciones. Los elementos implementados son explícitamente instanciados vía una especificación de XML (véase la Figura 1).

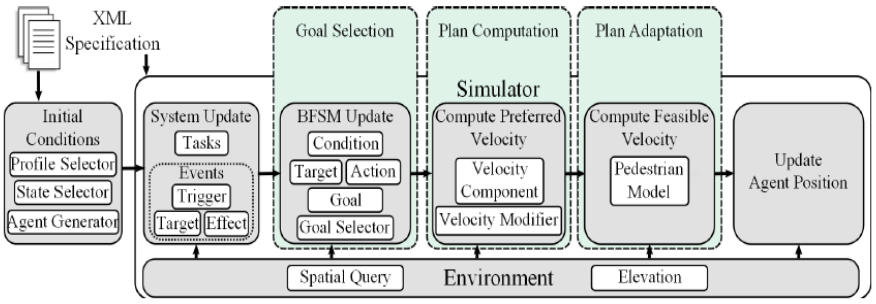

Fig. 1. La Arquitectura de Menge es una arquitectura modular, Fuente [13]. Como se aprecia XML es utilizado para establecer las condiciones iniciales y cómo se actualizará el sistema. La máquina de estado finito de comportamiento determina el comportamiento de cada agente, para el cual se establece su meta, se calcula su plan y se genera su modelo, el cual permite actualizar la posición del agente considerando un ambiente.

Como se aprecia en el siguiente código XML (véanse las Figuras 2 y 3) para la especificación de una escena se establece un experimento, en primer lugar, se define el tipo de consulta espacial que se llevará a cabo, el tiempo que se tomará la actualización, en este caso 0.1, se establecen también los modelos que se utilizarán en la simulación (en este caso los de Helbing y Karamouzas) y sus respectivos parámetros de ajuste.

Así mismo se determina el perfil del agente para el grupo 1, sus parámetros de ajuste: ángulo máximo de velocidad, máximo número de vecinos, obstáculos, tiempo de reacción, fuerza del cuerpo, fricción y fuerza de distancia), sus propiedades estadísticas (media y desviación estándar), se establecen también la masa para el modelo de Helbing, y el espacio personal y nivel de anticipación para el modelo de Karamouzas.

Más adelante, se establecen los parámetros para el grupo de agentes (Grupo 1), en los que se establece la grid (rejilla), la dirección, la densidad, el ancho, la población, la rotación, la distancia y el desplazamiento promedio, así como su desviación estándar. Finalmente, se establece el tipo de obstáculo -en este caso cerrado-, así como los vértices del mismo.

El código XML presentado en las Figuras 2 y 3 muestran la especificación de una Máquina de Estados Finitos de Comportamiento (BFSM) en Menge, en ella se establecen las metas, los posibles estados Caminar y Meta alcanzada, así como sus parámetros de operación. Se definen también las transiciones posibles: de caminar a la meta alcanzada, y de la meta alcanzada a caminar.

```
<Experiment version="2.0">
  <SpatialQuery type="kd-tree" test_visibility="false" />

  <Common time_step="0.1" />
  <Helbing agent_scale="2000" obstacle_scale="4000" reaction_time="0.5" body_force="1200"
      friction="2400" force_distance="0.015" />
  <Karamouzas orient_weight="0.8" fov="200" reaction_time="0.4" wall_steepness="2"
      wall_distance="2" colliding_count="5" d_min="1" d_mid="8" d_max="10" agent_force="4" />

  <AgentProfile name="group1" >
    <Common max_angle_vel="360" max_neighbors="10" obstacleSet="1" neighbor_dist="5" r="0.19"
        class="2" pref_speed="1.04" max_speed="2" max_accel="5" priority="0.0">
      <Property name="pref_speed" dist="n" mean="1.3" stddev="0.15" />
    </Common>
    <Helbing mass="80" />
    <Karamouzas personal_space="0.69" anticipation="8" />
    <ORCA tau="3.0" tauObst="0.15" />
  </AgentProfile>

  <AgentGroup>
    <ProfileSelector type="const" name="group1" />
    <StateSelector type="const" name="Walk" />
    <Generator type="hex_lattice" anchor_x="1.5" anchor_y="0.0" alignment="center" row_direction="y"
        density="1.8" width="4.0" population="100" rotation="-90" displace_dist="n"
        displace_mean="0.1" displace_stddev="0.03" />
  </AgentGroup>

  <ObstacleSet type="explicit" class="1">
    <Obstacle closed="1" >
      <Vertex p_x="-20" p_y="2.0" />
      <Vertex p_x="20" p_y="2.0" />
      <Vertex p_x="20" p_y="-2" />
      <Vertex p_x="-20" p_y="-2" />
    </Obstacle>
  </ObstacleSet>
</Experiment>
```

Fig. 2. Especificación de una escena para un corredor periódico [13].

```
<BFSM>
  <GoalSet id="0">
    <Goal type="AABB" id="0" min_x="-20" max_x="-15" min_y="-2.0" max_y="2"/>
  </GoalSet>

  <State name="Walk" final="0" >
    <GoalSelector type="explicit" goal_set="0" goal="0" />
    <VelComponent type="goal" />
  </State>
  <State name="GoalReached" final="0">
    <GoalSelector type="identity" />
    <VelComponent type="zero" />
    <Action type="teleport" dist="u" min_x="13.5" max_x="14" min_y="-1.5" max_y="1.5" />
  </State>

  <Transition from="Walk" to="GoalReached" >
    <Condition type="AABB" min_x="-40" max_x="-13.5" min_y="-2.0" max_y="2.0" inside="1" />
  </Transition>
  <Transition from="GoalReached" to="Walk" >
    <Condition type="auto" />
  </Transition>
</BFSM>
```

Fig. 3. Especificación del comportamiento para un corredor periódico [13].

FlameGPU. El modelado basado en Agentes es una aproximación que de manera intrínseca implica paralelismo, sin embargo, muchos de los ambientes de trabajo existentes fallan para explotarlo y frecuentemente se basan en algoritmos altamente serializados que manipulan agentes discretos móviles. Lo cual tiene serias implicaciones en la escala de los modelos y la velocidad en la que ellos pueden ser simulados. El propósito de FlameGPU es atender estas limitaciones del MBA (modelado basado en agentes) al utilizar una arquitectura de alto desempeño basada

en unidades de procesamiento gráfico GPUs (Graphical Processing Units). El ambiente de trabajo está diseñado con el paralelismo en mente, de manera que permita a los modelos de agentes escalar a tamaños masivos y que permita correr simulaciones dentro de restricciones de tiempo razonables. Adicionalmente a esta visualización, ésta se alcanza como datos que se mantienen directamente en la memoria del GPU donde esta puede ser renderizada directamente [7]. Técnicamente el Framework FlameGPU no es un simulador, es un ambiente de simulación basado en un template que mapea las descripciones formales de los agentes en código de simulación [7].

FlameGPU usa un script de función, para ello este es definido en un número de archivos de Función de Agentes. Los modelos de simulación se especifican utilizando un formato llamado X-Machine Mark-up Language (XMML) el cual tiene la sintaxis en XML que cuenta con esquemas que gobiernan el contenido.

Un modelo típico de una XMML consiste de la definición de un número de agentes en máquinas X, que incluyen información acerca del estado y la memoria, así como las funciones de transición de los agentes, un número de tipos de mensajes (cada uno de los cuales tiene una lista de mensajes globalmente accesibles) y un conjunto de capas de simulación que determinan el orden de ejecución de las funciones del agente lo que constituye una iteración de simulación simple [7], véase la Figura 4.

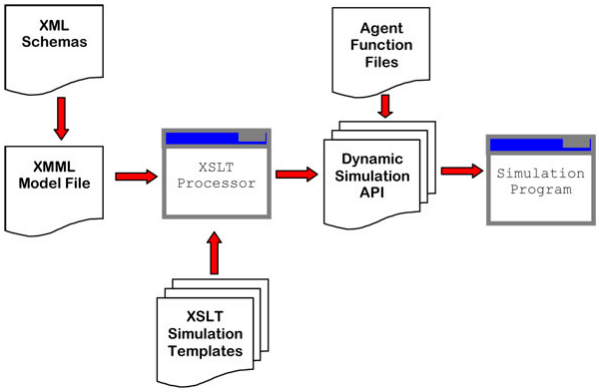

Fig. 4. Proceso de Modelado y Simulación en FlameGPU [7].

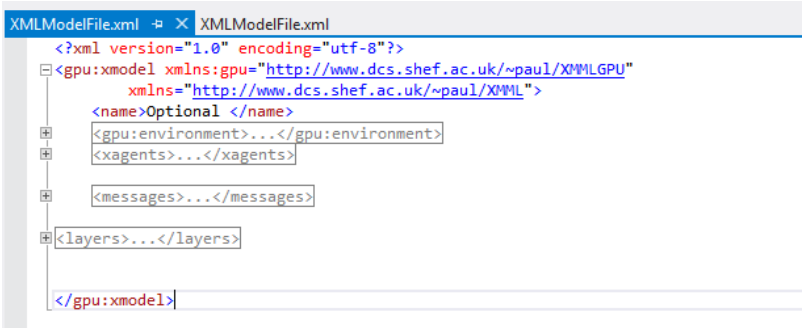

Fig. 5. Modelado en FlameGPU fuente propia basado en [7].

Dado una definición de un modelo en una máquina X, la generación de código se logra a través de transformaciones (XLST - Extensible Stylesheet Transformations, o transformaciones de estilo de hojas extensibles), a partir de ella se generan los templates para la simulación dinámica en una Aplicación (API) a partir de la cual se genera el programa de simulación.

En la Figura 4, se muestra la estructura XML para la especificación de un modelo en FlameGPU, primeramente se determina que se utilizaran GPUs, el nombre del modelo, el ambiente de trabajo de GPUs junto con todas sus especificaciones, los contenidos de los Agentes X (X-agents), los mensajes y las capas de aplicación.

En la Tabla 2 se muestra nuestra propuesta diseñada para la integración de estos esquemas, la cual hemos denominado XBFSM (eXtended Behavior Finite State Machine) o bien Máquina de Estado Finito con comportamiento extendido.

Tabla 2. Comparativo entre los esquemas XML de Menge, FlameGPU y nuestra propuesta.

Elemento	Menge	FlameGPU	Propuesta
Máquina de estado finito	BFSM	X-Agent	XBFSM
Definición de escenas en XML	<Experiment> <SpatialQuery …/ > <model> … </model>	<gpu model …> </name> … </name> <gpu:environment> … </gpu:environment>	<Experiment> <SpatialQuery …/ > <name>…</name> <gpu:environment> … </gpu:environment>
<AgentProfile> … </AgentProfile> <AgentGroup> <Obstacle> </Obstacle> </Agentgroup>	<Xagents> … </Xagents> <messages> … </messages> <layers> … </layers>	<Xagents> <AgentProfile> </AgentProfile> <Agentgroup> <messages> <layers></layers> </Agentgroup> </Xagents>	<AgentProfile> … </AgentProfile> <AgentGroup> <Obstacle> </Obstacle> </Agentgroup>
Creación de agentes	<AgentProfile> … </AgentProfile> <AgentGroup> <Obstacle> </Obstacle> </Agentgroup>	<Xagents> … </Xagents> <messages> … </messages> <layers> … </layers>	<Xagents> <AgentProfile> </AgentProfile> <Agentgroup> <messages> <layers></layers> </Agentgroup> </Xagents>
Final de la escena	</Experiment>	</gpu:xmodel>	</Experiment>

En esta tabla, la primera columna identifica el elemento analizado y las columnas restantes los esquemas XML para Menge, FlameGPU y XBFSM respectivamente.

Aunque se muestran tres secciones, cabe señalar que estas forman parte de la misma definición de los esquemas XML, la primera sección corresponde al inicio de

la definición de escenas. En Menge se llama experimento, en FlameGPU se denomina modelo, en la XBFSM, se llama experimento, e incluye la consulta espacial para identificar la posición del agente, el nombre y si establecerá un ambiente GPU o no en el experimento, cabe señalar que para pocos agentes este ambiente debe removerse o declararse como nula, para cantidades importantes de agentes esta definición es esencial para la realización del experimento en tiempos razonables.

La sección denominada creación de agentes, en Menge se define el perfil del agente, el comportamiento del grupo y los obstáculos, en FlameGPU se especifica el funcionamiento de los agentes X, los mensajes y capas que se utilizan en la simulación. En la propuesta consolidada se incorpora dentro de los agentes X, el perfil del agente, el funcionamiento del grupo, los mensajes y las capas, cabe señalar que en estas últimas se definen los obstáculos requeridos en Menge.

Finalmente, en el último renglón se define el final del esquema XML, para Menge mediante la etiqueta </Experiment>, para FlameGPU </gpumodel>, y para la XBFSM </Experiment>.

4. Metodología

La metodología propuesta para probar los esquemas XML arriba descritos es la siguiente:

1. Si el número de agentes a simular es pequeño se utiliza el esquema XML de Menge, si el número de agentes a simular es grande se recomienda utilizar FlameGPU. Al emplear nuestra propuesta solo se deberá incluir la definición del ambiente GPU (<GpuEnvironment> </GpuEnvironment>).
2. Se realiza la simulación en los software de prueba (en Menge y FlameGPU) utilizando los esquemas propuestos.
3. Se enlaza la salida de la herramienta y se visualiza en Unity3D.
4. Se evalúan los resultados.

Hardware. Estación de trabajo Dell 7450, 6 Gigabytes de Memoria RAM, Disco Duro 1.3 Terabytes. 2 procesadores XEON.; 1 Tarjeta NVIDIA Tesla C2075.

Software. Microsoft Visual Studio Community 2013 [19], FlameGPU Versión 1.4 [7] para CUDA 7.0 [2]; Menge [13] y el Software de simulación Unity3D edición personal [18].

5. Resultados y discusión

5.1. Simulación con Menge y Unity3D

En la Figura 6 se muestra el resultado de la simulación del demo 4square de Menge, el cual fue descompuesto en sus componentes descritos en los esquemas XML, simulación y visualización en tiempo real como se describe en [3].

5.2. Simulación con FlameGPU

La figura 7 muestra la ejecución de Decenas de miles de agentes (caminantes) en un edificio, el visualizador permite analizar la dinámica de la simulación, la ejecución de esta simulación requiere de una tarjeta NVIDIA GPU con capacidades de procesamiento 2.0 o superior para su adecuado funcionamiento.

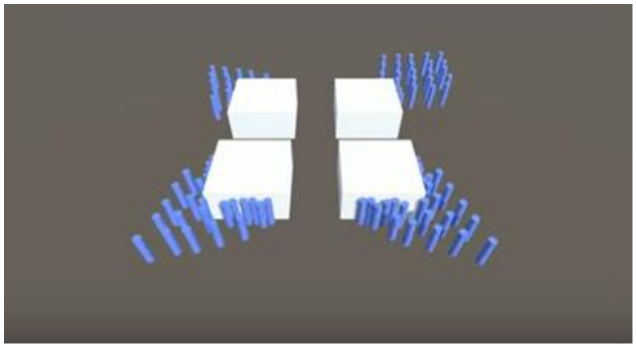

Fig. 6. Simulación del demo 4Square de Menge en Unity3D, basado en [3].

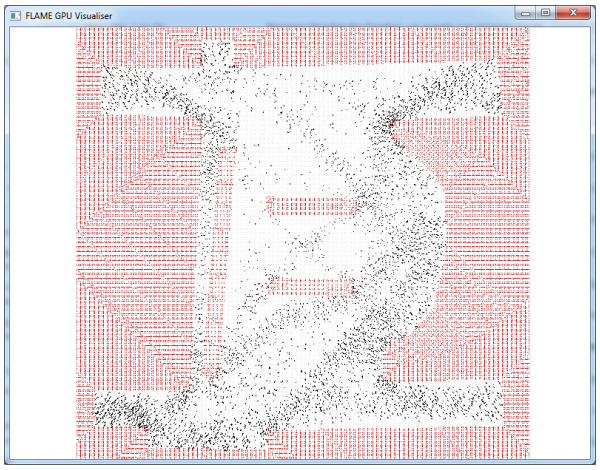

Fig. 7. Simulación del demo de Caminantes de FlameGPU en su visualizador (fuente propia obtenida de la operación de FlameGPU).

6. Discusión

La simulación de multitudes es implementada mediante agentes que emplean máquinas de estados finitas, estas utilizan XML para crear escenas, y definir el comportamiento de los agentes. Para el caso de Menge se utiliza una máquina de estados finitos con comportamiento (BFSM), para FLAMEGPU se utiliza un modelo para agentes denominado X-Agent. Al utilizar Unity3D la calidad y el efecto visual de la simulación realizada en 2D, tanto en Menge como en FLAMEGPU, es mejor y

abre la posibilidad a la realización de simulaciones más realistas. La fortaleza principal de la BFSM es la posibilidad de incorporar nuevos módulos que mejoren la simulación, mientras que la fortaleza más grande del X-Agent es la definición de etiquetas para explotar el uso de las GPU. El nuevo esquema XML propuesto integra estas dos fortalezas en lo que demos denominado eXtended Behavior Finite State Machine (XBFSM).

Las máquinas de estado finito son una herramienta muy poderosa para representar y modelar comportamientos para diversos tipos de agentes dentro de las simulaciones, ya que permiten ver de forma clara cuál es la condición actual del agente y que eventos o condiciones disparan las transiciones entre estados, sin embargo, esta aproximación dista mucho de ser óptima en el manejo de multitudes por varios factores. Las multitudes están compuestas por un conjunto de agentes individuales que deben comportarse de forma distinta en cualquiera de las situaciones que la simulación presente. Utilizando un esquema tradicional de máquinas de estado finito estamos limitados a un conjunto de estados y condiciones que se activan con parámetros definidos, por lo tanto utilizar este esquema con un grupo grande de agentes implica grandes esfuerzos para ajustar el sistema y de esta manera conseguir comportamientos aproximados a la realidad.

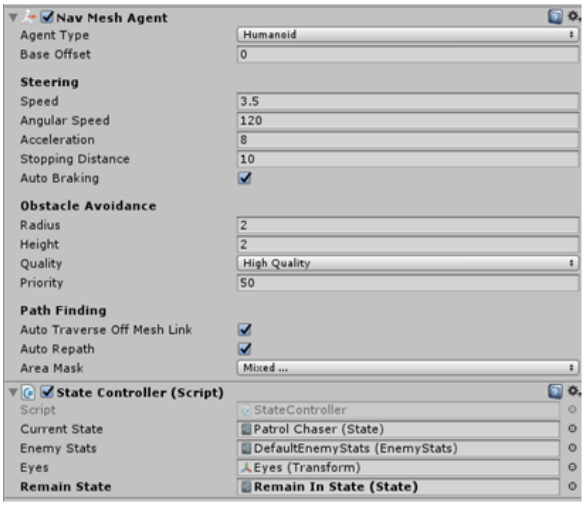

Fig. 8. Parámetros que pueden ser ajustados en la simulación desde el editor y una construcción simple de las máquinas de estado finito.

Para la simulación en Unity proponemos una modificación al manejo de las máquinas de estado finito, de forma que no es necesario hacer cambios manualmente o modificar las máquinas de cada agente para que se ajuste al perfil individual que se pretende simular. Nuestra simulación cuenta con una librería común de conocimiento en la cual los agentes pueden consultar los parámetros generales de la simulación, por ejemplo: velocidad, rango de visión y velocidad de giro por mencionar algunos, y dependiendo de su perfil se ajustan individualmente de forma automática; de esta

manera personajes que pertenezcan por ejemplo al grupo de "niños", tendrán distintas capacidades a los que pertenezcan al grupo de "adultos". Sin embargo ajustar parámetros no resuelve en la totalidad el problema de la individualidad y comportamientos distintos en los agentes, es por eso que también podemos ajustar las condiciones que disparan la transición de los eventos o incluso que en la misma máquina de estados los agentes reaccionen de forma distinta, por ejemplo en una situación de emergencia dos agentes en estado de pánico pueden tener reacciones totalmente opuestas, mientras que uno busca escapar el otro puede decidir intentar localizar a los miembros de un grupo determinado o seguir a otro agente. La figura 8 muestra cómo podemos ajustar y seleccionar esos parámetros desde el editor de Unity, o hacerlos de manera automática utilizando un script.

7. Conclusiones y trabajo futuro

Para el software analizado se identificó que para la realización de simulaciones se requieren de al menos un esquema XML que defina las escenas, y de otro esquema para determinar el comportamiento del agente y el de su grupo. Si se desean simular algunas centenas o algunos miles de usuarios la opción más adecuada es Menge, por otro lado, si lo que se desea es modelar decenas de miles, cientos de miles o más agentes la opción más adecuada es FlameGPU.

Contribución. En esta investigación se presenta una nueva propuesta para la generación de máquinas de estados finitos, denominada XBFSM (eXtended Behavior Finite State Machine) que permite integrar los esquemas XML de Menge y FlameGPU, al incluir dentro de su definición lo mejor de cada uno de ellos.

Limitaciones: Se requiere de un middleware que permita transformar el esquema XML de nuestra propuesta al esquema XML nativo de cada aplicación (Menge o FlameGPU). Este trabajo de ninguna forma está completo, es nuestro deseo realizar simulaciones diversas, entre las que se incluyen la simulación de multitudes en situaciones de emergencia y desastres, como sismos e incendios y probarlas en instituciones educativas en México utilizando Menge y FlameGPU, probar con distinto número de usuarios (densidad de población) y llevar a cabo estudios comparativos de desempeño. Es de particular importancia la preparación de reservas en caso de desastre, para ello se utilizarán como base las investigaciones de [1] al respecto. Así mismo se buscará mejorar el algoritmo de desplazamiento que utiliza la máquina de estados finitos mediante el uso de algoritmos bioinspirados.

Referencias

1. Campbell, A.M., Jones, P.C.: Prepositioning supplies in preparation for disasters. European Journal of Operational Research 209(2), pp.156–165 (2011)
2. CUDA Homepage, CUDA Toolkit 7.0, https://developer.nvidia.com/cuda-toolkit-70, last accessed 2018/04/04.
3. Curtis, S.: Menge en Unity, https://www.youtube.com/watch?v=RpkL6HqqMvU (2018)

4. De Gyves, O., Toledo, L., Rudomín, I.: Comportamientos en simulación de multitudes: revisión del estado del arte. Research in Computer Science, 62, pp. 319–334 (2013)
5. Helbing, D., Buzna, L., Johansson, A., Werner, T.: Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. Transportation Science, 39(1), pp. 1–24 (2005)
6. Karbovskii, V., Voloshin, D., Karsakov, A., Bezgodov, A., Gershenson, C.: Multimodel agent-based simulation environment for mass-gatherings and pedestrian dynamics. Future Generation Computer Systems, 79, pp. 155–165 (2018)
7. FlameGPU, Flexible Large Scale Environment Agent Mobile Environment for the GPU. http://www.flamegpu.com/downloads/links (2018)
8. Liu, H., Liu, B., Zhang, H., Li, L., Qin, X., Zhang, G.: Crowd evacuation simulation approach based on navigation knowledge and two-layer control mechanism. Information Sciences, 436, pp. 247–267 (2018)
9. Löhner, R.: On the modeling of pedestrian motion. Applied Mathematical Modelling, 34(2), pp. 366–382 (2010)
10. Luo, L., Chai, C., Ma, J., Zhou, S., Cai, W.: ProactiveCrowd: Modelling Proactive Steering Behaviours for Agent-Based Crowd Simulation. In: Computer Graphics Forum. 37(1), pp. 375–388. The Eurographics Association and John Wiley & Sons Ltd (2018)
11. Martínez-Romero, A.: Simultan: un Sistema de I.A. per a Simulació de Multituds. Tesis Grau en Enginyeria en Informàtica Especialitzat en Computació. Universitat Politècnica de Catalunya. Facultat d'Informàtica de Barcelona (2016)
12. Curtis, S., Best, A., Manocha, D.: Menge: A modular framework for simulating crowd movement. Collective Dynamics, 1, pp. 1–40 (2016)
13. Menge: What is Menge? http://gamma.cs.unc.edu/Menge/ (2018)
14. Millan, E., Hernández, B., Rudomín, I.: Large crowds of autonomous animated characters using fragment shaders and level of detail. In: ShaderX5: Advanced Rendering Techniques, chap. Beyond Pix, pp. 501–510 (2006).
15. Patent US7596745B2: Programmable hardware finite state machine for facilitating tokenization of an XML document (2018)
16. Rudomín, I., Millán, E., Hernández, B.: Fragment shaders for agent animation using finite state machines. Simulation Modelling Practice and Theory, 13(8), pp. 741–751 (2005)
17. Rudomín, I., Millán, E., Hernández, B., Díaz, M., Rivera, D.: Art applications for crowds. The Knowledge Engineering Review, 23(4), pp. 399–412 (2008)
18. Unity3D: Software de simulación Unity3D edición personal (2018)
19. Visual Studio 2013: Visual Studio Community 2013 Release Notes, https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2013-community-vs (2018)