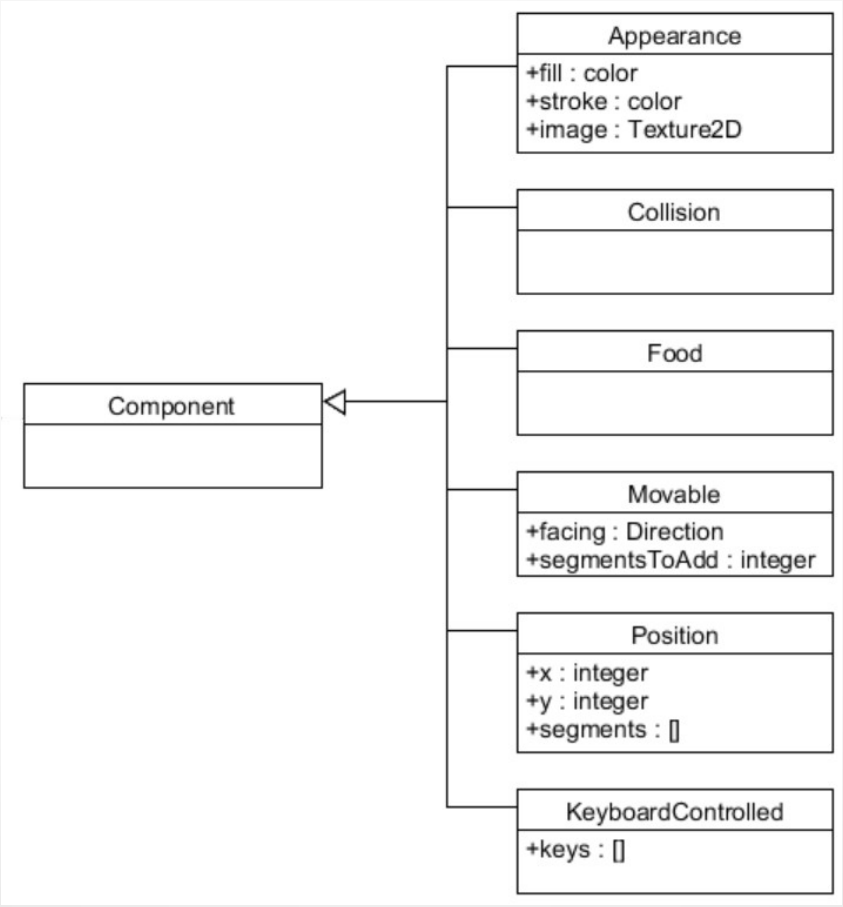
ECS Example : Snake Mini-Game

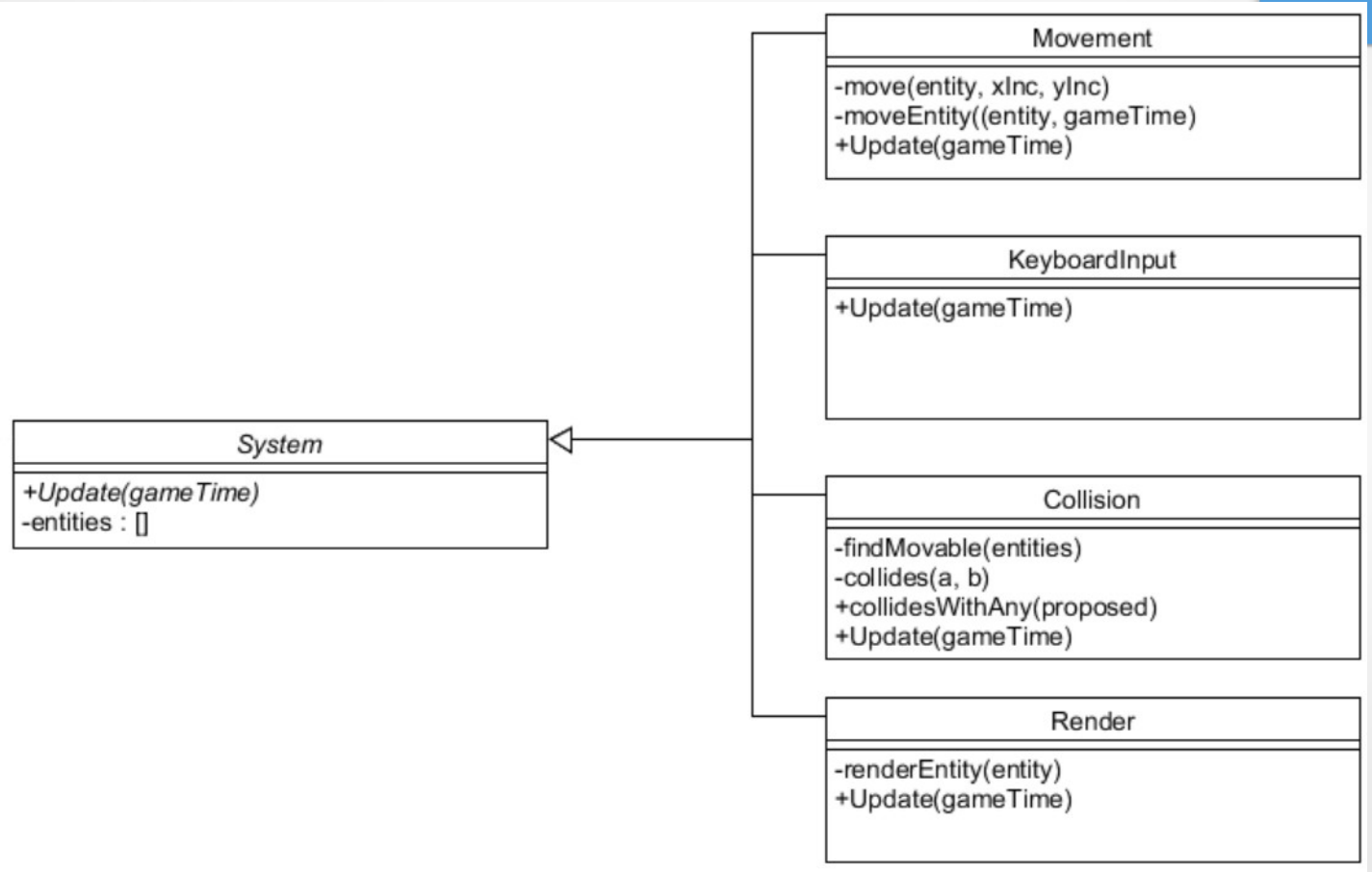
Components


Appearance
+fill : color
+stroke : color
+image : Texture2D
Collision
Food
Component
Movable
+facing : Direction
+segmentsToAdd : integer
Position
+x : integer
+y : integer
+segments : []
KeyboardControlled
+keys : []

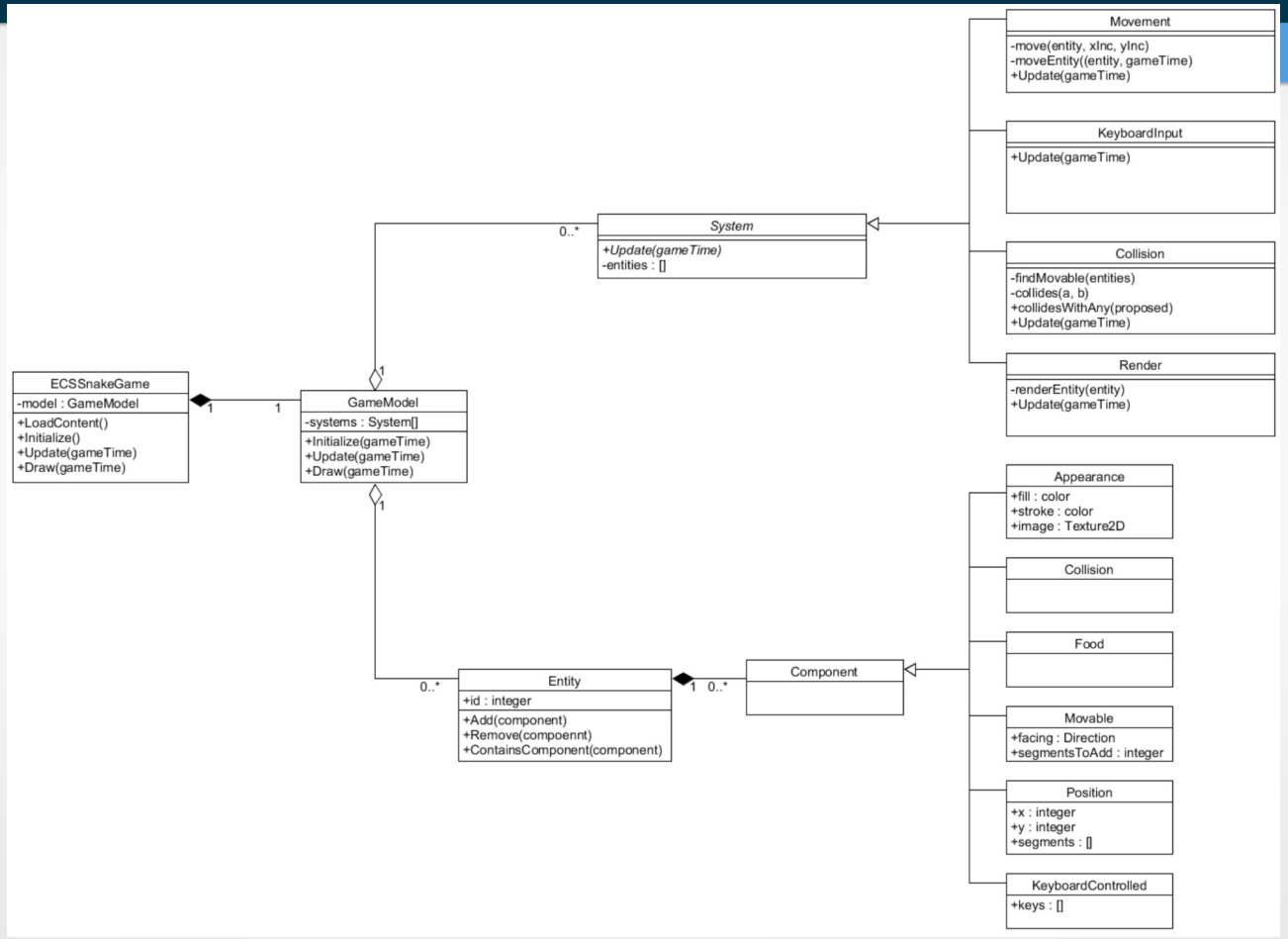
Entities

- Border : { Appearance, Position, Collision }
- Obstacle : { Appearance, Position, Collision }
- Food : { Appearance, Position, Collision, Food }
- Snake : { Appearance, Position, Collision, Movable, KeyboardControlled }

Systems


Movement
-move(entity, xInc, yInc)
-moveEntity((entity, gameTime)
+Update(gameTime)
KeyboardInput
+Update(gameTime)
System
+Update(gameTime)
-entities : []
Collision
-findMovable(entities)
-collides(a, b)
+collidesWithAny(proposed)
+Update(gameTime)
Render
-renderEntity(entity)
+Update(gameTime)

Overview


Movement
-move(entity, xInc, yInc)
-moveEntity((entity, gameTime)
+Update(gameTime)
KeyboardInput
+Update(gameTime)
System
+Update(gameTime)
-entities : []
0..*
Collision
-findMovable(entities)
-collides(a, b)
+collidesWithAny(proposed)
+Update(gameTime)
Render
-renderEntity(entity)
+Update(gameTime)
ECSSnakeGame
-model : GameModel
+LoadContent()
+Initialize()
+Update(gameTime)
+Draw(gameTime)
1
1
GameModel
-systems : System[]
+Initialize(gameTime)
+Update(gameTime)
+Draw(gameTime)
1
1
Appearance
+fill : color
+stroke : color
+image : Texture2D
Collision
Food
Movable
+facing : Direction
+segmentsToAdd : integer
Position
+x : integer
+y : integer
+segments : []
KeyboardControlled
+keys : []
0..*
Entity
+id : integer
+Add(component)
+Remove(compoennt)
+ContainsComponent(component)
1
0..*
Component

Components

```
public class Appearance : Component
{
	public Texture2D image;
	public Color fill;
	public Color stroke;

	public Appearance(Texture2D image, Color fill, Color stroke)
	{
		this.image = image;
		this.fill = fill;
		this.stroke = stroke;
	}
}
```

```
public class Position : Component
{
	public List<Point> segments = new List<Point>();
	public int x { get { return segments[0].X; } }
	public int y { get { return segments[0].Y; } }

	public Position(int x, int y)
	{
		segments.Add(new Point(x, y));
	}
}
```

```
public class Collision : Component
{
}
```

Entity

```
public sealed class Entity
{
	private readonly Dictionary<Type, Component> components = new Dictionary<Type, Component>();
	private static uint m_nextId = 0;

	public Entity()
	{
		Id = m_nextId++;
	}

	public uint Id { get; private set; }

	public bool ContainsComponent(Type type)
	{
		return components.ContainsKey(type) && components[type] != null;
	}

	public bool ContainsComponent<TComponent>()
		where TComponent : Component
	{
		return ContainsComponent(typeof(TComponent));
	}
	...
}
```

Entity

```
public sealed class Entity
{
	private readonly Dictionary<Type, Component> components = new Dictionary<Type, Component>();
	private static uint m_nextId = 0;

	public Entity()
	{
		Id = m_nextId++;
	}

	public uint Id { get; private set; }

	public bool ContainsComponent(Type type)
	{
		return components.ContainsKey(type)
	}

	public bool ContainsComponent<TComponent>
		where TComponent : Component
	{
		return ContainsComponent(typeof(TC
	}
	...
}
```

```
public sealed class Entity
{
	...

	public void Add(Component component)
	{
		this.components.Add(component.GetType(), component);
	}

	public void Remove<TComponent>()
		where TComponent : Component
	{
		this.components.Remove(typeof(TComponent));
	}

	public TComponent GetComponent<TComponent>()
		where TComponent : Component
	{
		return (TComponent)this.components[typeof(TComponent)];
	}
}
```

Creating an Obstacle

```
public class Obstacle
{
      public static Entity create(Texture2D square, int x, int y)
      {
            var obstacle = new Entity();

            obstacle.Add(new Components.Appearance(square, new Color(0, 255, 0), Color.Black));
            obstacle.Add(new Components.Position(x, y));
            obstacle.Add(new Components.Collision());

            return obstacle;
      }
}
```

Creating a Snake

```
public class SnakeSegment
{
	private const int MOVE_INTERVAL = 150; // milliseconds
	public static Entity create(Texture2D square, int x, int y)
	{
		var snake = new Entity();

		snake.Add(new Components.Appearance(square, Color.White, Color.Black));
		snake.Add(new Components.Position(x, y));
		snake.Add(new Components.Collision());
		snake.Add(new Components.Movable(Components.Direction.Stopped, MOVE_INTERVAL));
		snake.Add(new Components.KeyboardControlled(
			new Dictionary<Keys, Components.Direction>
			{
				{ Keys.Up, Components.Direction.Up },
				{ Keys.Down, Components.Direction.Down },
				{ Keys.Left, Components.Direction.Left },
				{ Keys.Right, Components.Direction.Right }
			}));

		return snake;
	}
}
```

Implementing a System

...show code in editor...

Game Model (game loop) – Update systems

```
public void Update(GameTime gameTime)
{
	m_sysKeyboardInput.Update(gameTime);
	m_sysMovement.Update(gameTime);
	m_sysCollision.Update(gameTime);

	foreach (var entity in m_removeThese)
	{
		RemoveEntity(entity);
	}
	m_removeThese.Clear();

	foreach (var entity in m_addThese)
	{
		AddEntity(entity);
	}
	m_addThese.Clear();
}
```

Game Model (game loop) – Update systems

```
public void Draw(GameTime gameTime)
{
    m_sysRenderer.Update(gameTime);
}
```