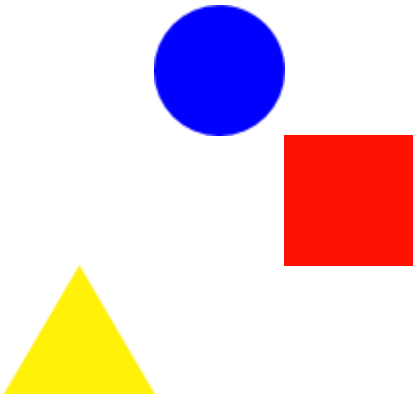


Decades Game Design
Document
Concept and Design 2



Working Title

bauhaus

Verbs

analyse, predict, plan,
capture, turn, block,
risk, react, discover,
survive.

High Level Concept

*Jump into a fast-paced
territorial battle that
demands speedy planning
and decisive action to get
ahead of the competition.*

Keywords

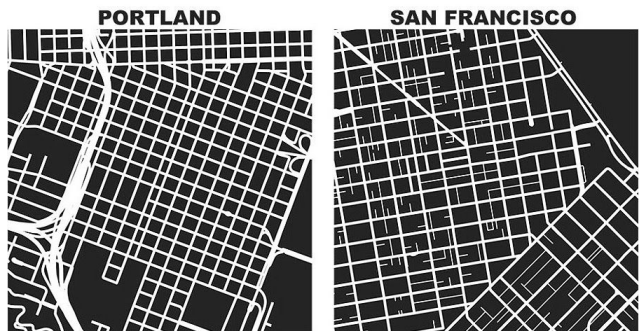
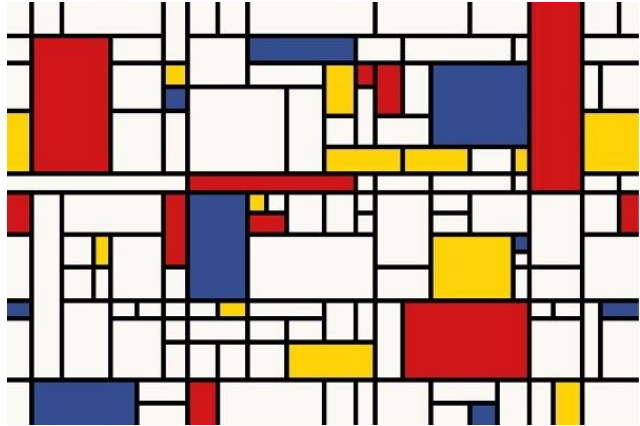
1920s, Bauhaus, Mondrian,
Minimalism, Creative, Art,
Territory, Game.

Research

Inspired by the apolitical philosophy and the bold and enigmatic visual styles of the Bauhaus movement, as well as preceding constructivist and modernist influences, we set out to design a game that is thematised by the art and design that is characteristic of this period. The artwork of Piet Mondrian was the primary catalyst for our game's theme and aesthetic style.

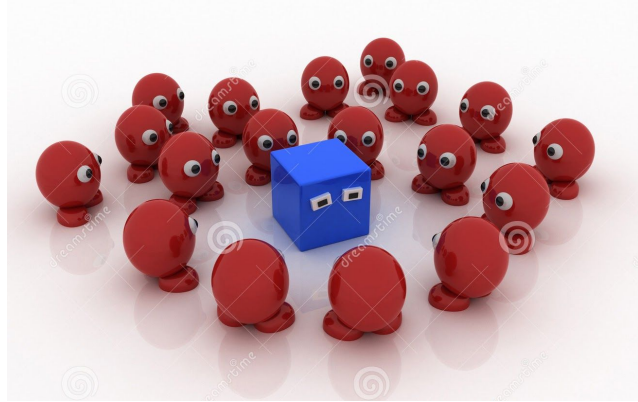
The origin of the Bauhaus movement is attributed to the teachings of a German art school that was founded in 1919 by Walter Gropius. Spanning approximately 14 years, the movement was hugely influential for subsequent developments in popular creative fields, giving rise to prominent artists like Piet Mondrian and Wassily Kandinsky.


Moodboard



Jeff Boring



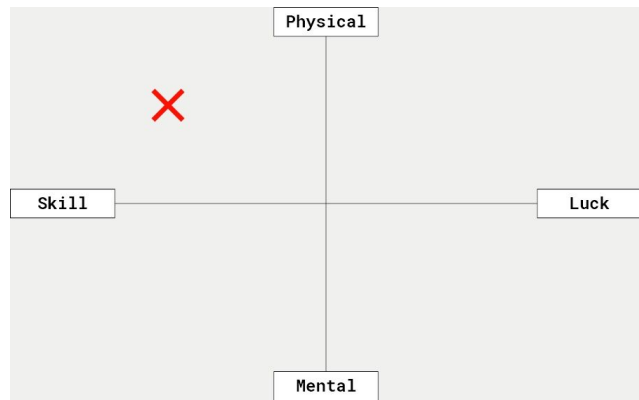




Concept & Influence

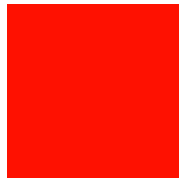
Guided by the requirement for a clear conceptual and thematic connection to our chosen decade, we decided to make use of the grid-like structures and bold primary colours that are prevalent in the work of Bauhaus painter Piet Mondrian. To this end, we have designed our game objects and mechanics in such a way that the game's overall composition loosely resembles one of Mondrian's paintings once the victory condition has been reached. This hidden resemblance will be unknown to the Player at first, gradually becoming more apparent as the Player progresses through the game. By controlling the game's aesthetic in this way, we have ensured that gameplay becomes as much an act of creation as one of competition.

Matrix

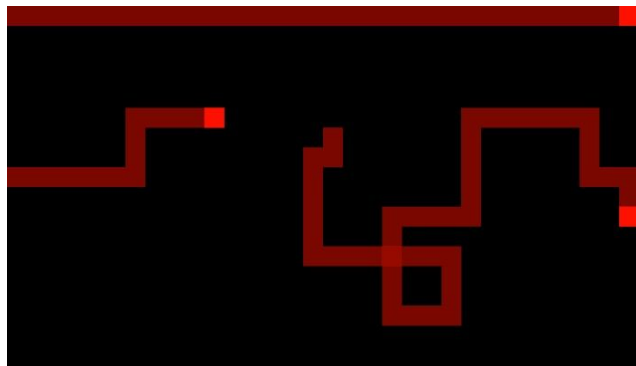


Concept Art

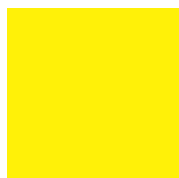
Player



Player Trail

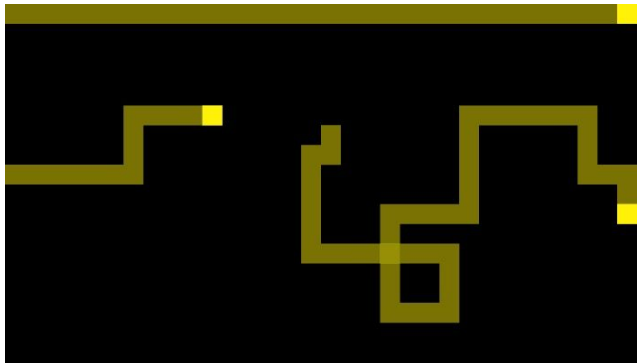
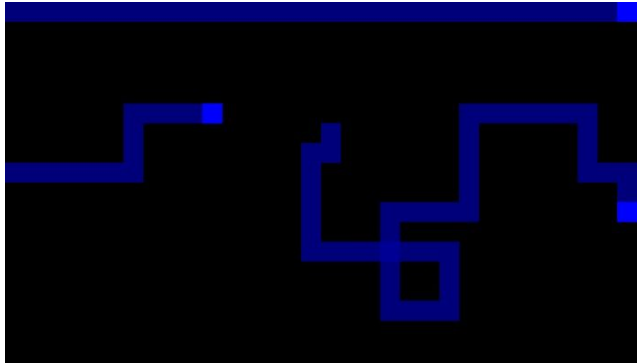


Enemies

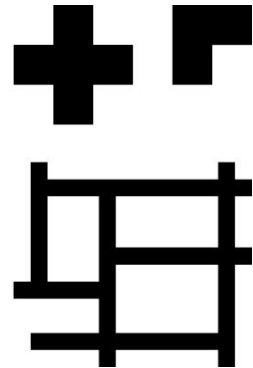
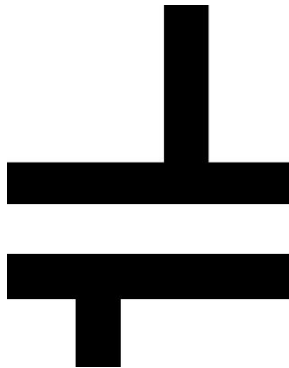


Concept Art cont.

Enemy Trails



Roads



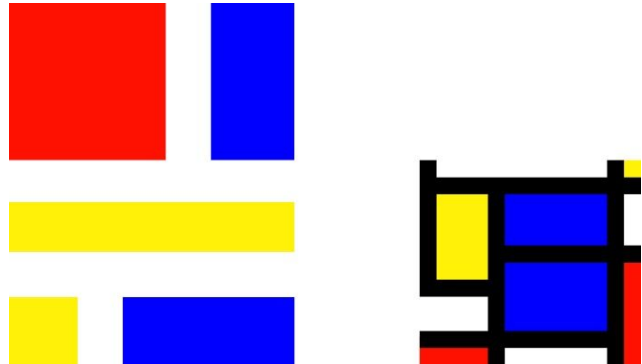
Concept Art cont.

Buildings

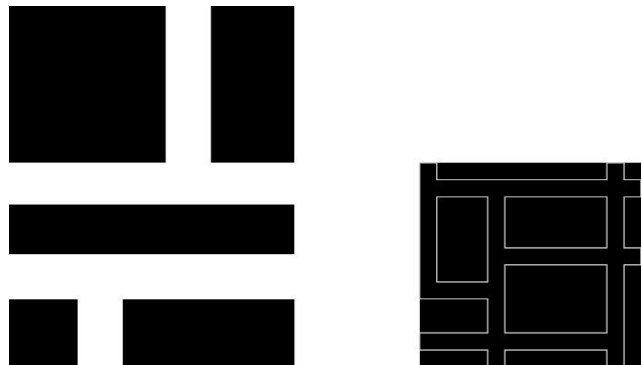
→ *neutral (white)*



→ *captured (coloured R/B/Y)*



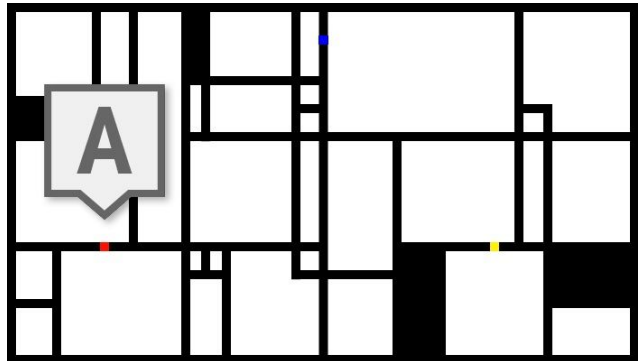
→ *locked (black)*



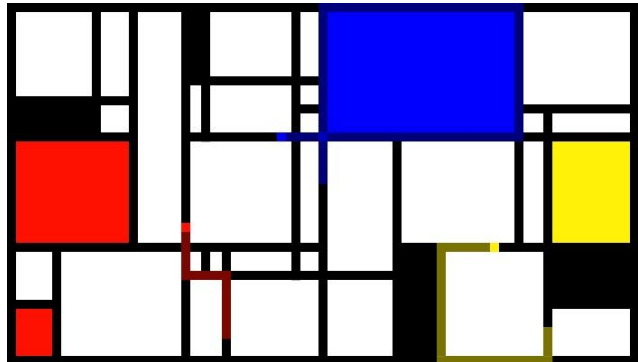
Concept Art cont.

Map

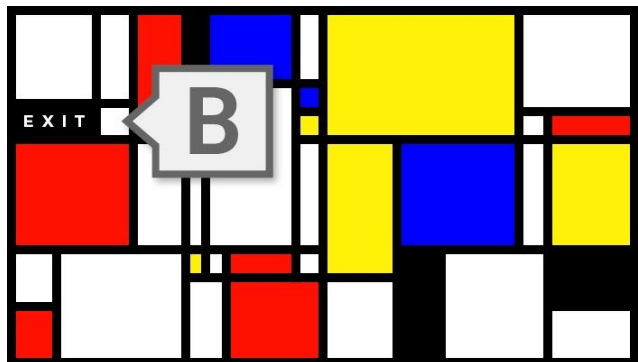
→ *initial*



→ *during gameplay*



→ *game over*



Exp. of Play

To further our thematic connection to Mondrian's work, we've designed the game's primary, secondary and tertiary mechanics to compliment each other in a way that echoes Mondrian's use of the primary colour triad. Successful gameplay outcomes are achieved by learning how to balance the three game mechanics, which are as follows:

Game Mechanics

Primary (Capturing)

Surround neutral city blocks to capture territory

Territory captured by Player becomes Player colour

Territory captured by an enemy becomes corresponding enemy colour

Surround enemy city blocks to recapture territory

Black city blocks are locked and cannot be captured

Exp. of Play cont.

Secondary (Turning)

When turning a corner, player ability is measured in three levels.

Making a “perfect” turn will increase Player speed and trail length.

Making a “good” turn has neither of these benefits.

Making a “crashed” turn will reset Player speed and trail length to initial values.

Tertiary (Obstructing)

Colliding with the Player trail resets enemy speed and trail length to initial values.

Colliding with an enemy trail resets Player speed and trail length to initial values.

Head-on collisions reset both Player and enemy speed and trail length to initial values.

Gameplay Summary

The Player will move around the field of play by making a series of rapid decisions that are based on multiple factors.

Exp. of Play cont.

The Player will learn to maintain an awareness of the positions and relative speeds of other objects and obstacles in the game. The Player must use this awareness to assess the level of risk that impending decisions may present. High-risk decisions can lead to high rewards and rapid gameplay progression if luck is on your side. High-risk decisions can also lead to the opposite, in some cases irreparably limiting the Player's chances of winning the game.

Player Strategy

Reaching a successful game outcome will require the Player to develop a balanced gameplay strategy that takes optimum advantage of all three game mechanics; Capturing, Turning and Obstructing.

Exp. of Play cont.

One approach could be to avoid the opponent as much as possible and focus on capturing a lot of territory quickly. The downside to this approach is that there is nothing to stop the opponent from capturing more territory than the Player. Another approach could be to focus on obstructing opponents to slow them down as much as possible. The downside here is that the Player will not gain any territory. A more balanced and thereby largely more successful approach would be to seek out and take advantage of opportunities that allow the Player to accomplish both of these objectives simultaneously. Choosing a path that will both obstruct an opponent and lead to a block capture would be a highly effective strategy.

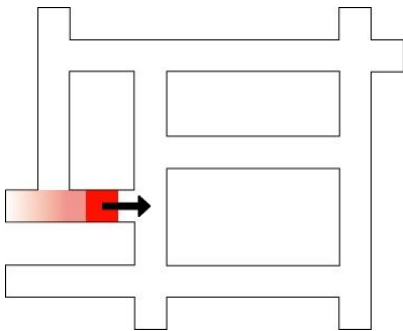


Storyboard

Player Controls

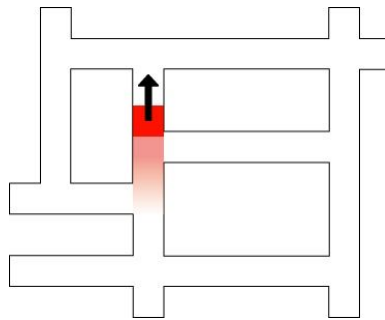
Player movement is controlled using the Up, Down, Left and Right keys (on Keyboard) or using Thumbstick (on Controller).

1.



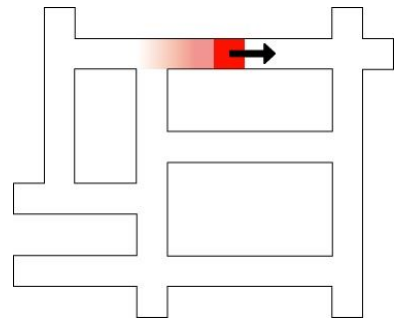
Rapidly approaching a corner

2.



Presses "Up" key to change direction

3.

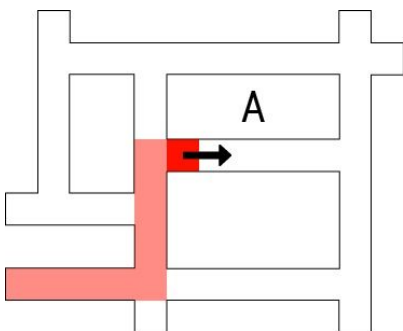


Presses "Right" key to change direction

Core Mechanics

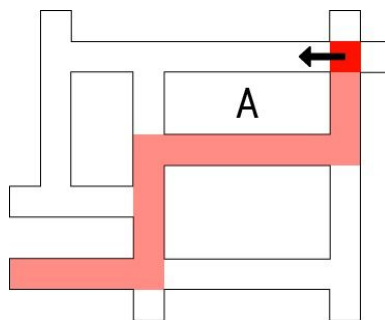
Primary Game Mechanic (Capturing Territory). When your trail is long enough you can start surrounding city blocks to capture territory.

1.



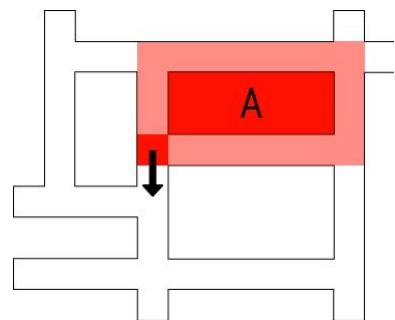
Plans to surround city block A

2.



Starts to surround city block A

3.

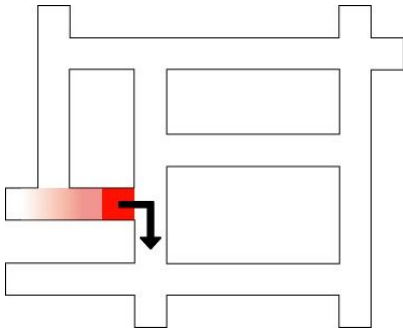


Once surrounded, city block A is captured

Storyboard cont.

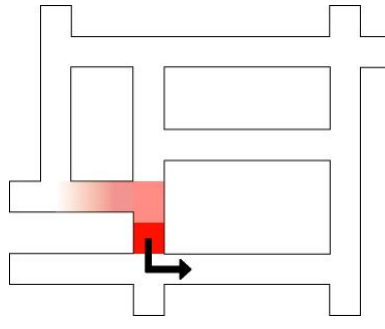
Secondary Game Mechanic (Turning Corners). Perfect timing on turns is rewarded with +1 Player speed and trail length.

1.



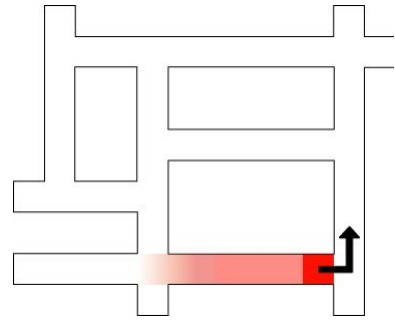
Performs perfect turn
Player Speed: 2
Trail length: 1

2.



Performs perfect turn
Player Speed: 3
Trail length: 2

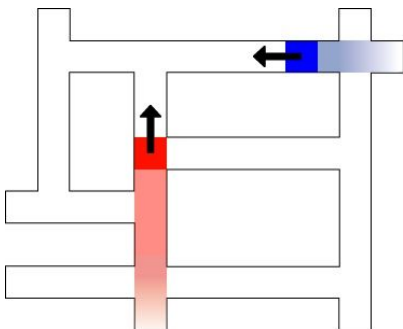
3.



Performs perfect turn
Player Speed: 4
Trail length: 3

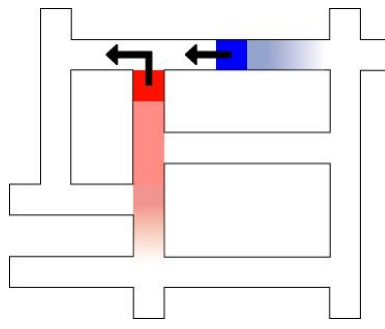
Tertiary Game Mechanic (Obstructing Opponents). Both the Player and the opponents can trap and be trapped.

1.



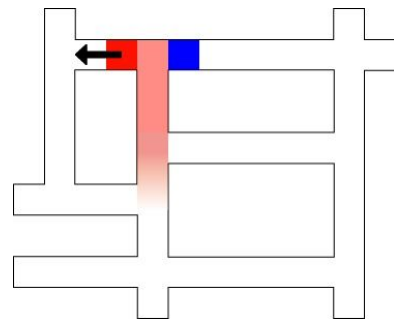
Red plans to obstruct

2.



Red reaches corner

3.



Blue's speed and trail is reset to 0

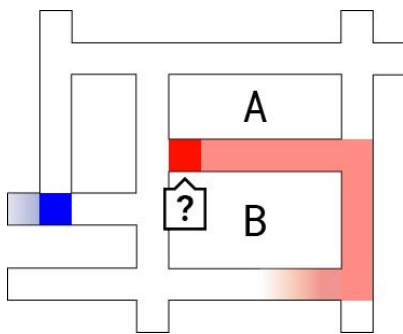
Storyboard cont.

Sample Challenge

A scenario that the Player may encounter during gameplay. Here the Player must assess the options and the corresponding risk/reward.

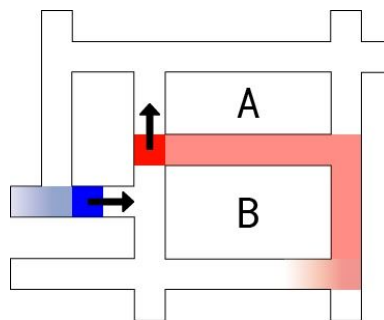
Scenario A (no risk)

1.



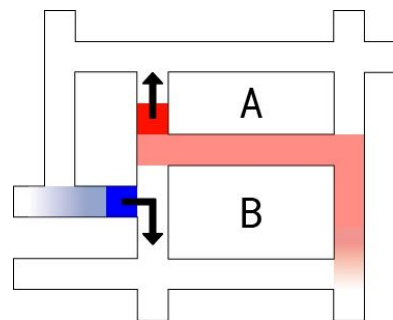
Player is attempting to capture block B. Blue opponent poses a potential risk.

2.



Player evades collision with blue. Capturing block B is no longer possible.

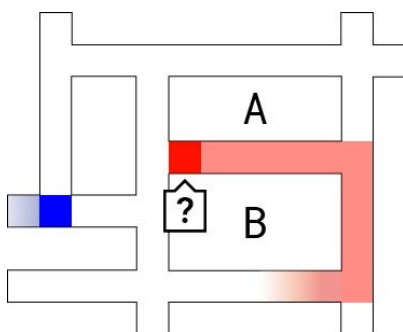
3.



Player will capture block A. Blue cannot stop this.

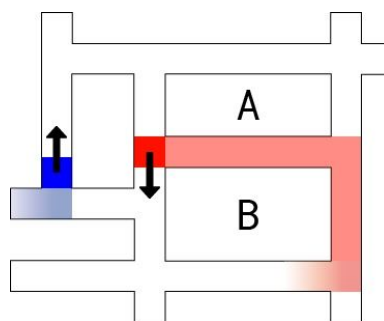
Scenario B (high risk)

1.



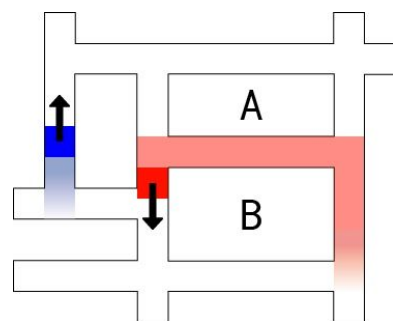
Player is attempting to capture block B. Blue opponent poses a potential risk.

2.



Player continues to capture block B, risking collision with Blue opponent.

3.



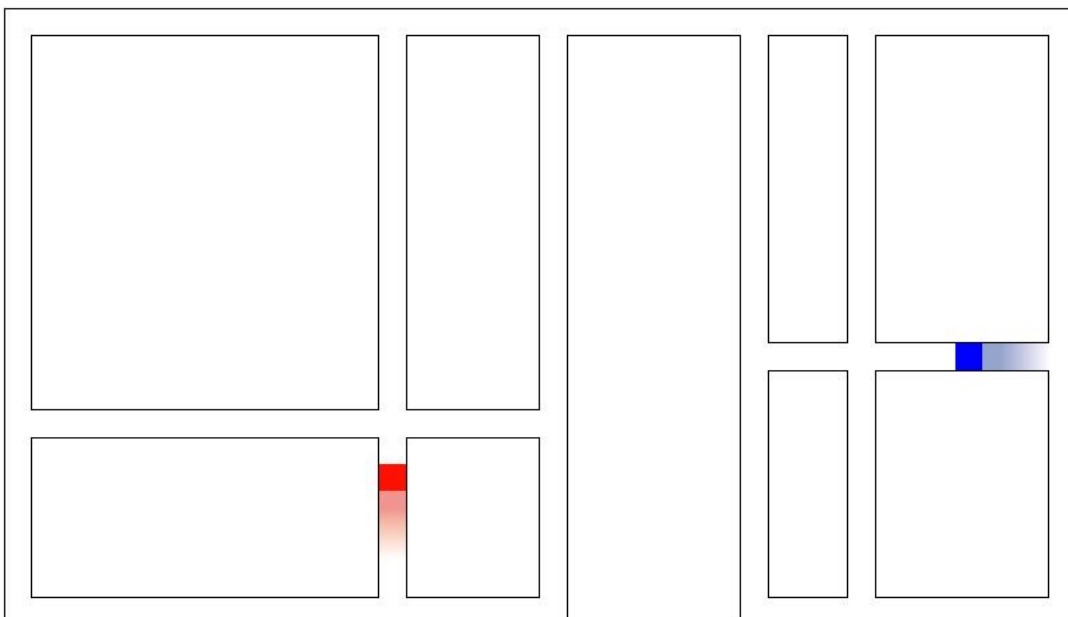
In this outcome the risk pays off. Player will capture block B.

Storyboard cont.

Game Progression

Early in the game the Player will see only a small area of the map. Gameplay will largely be composed of learning turning and capturing mechanics. Risk management and conflict with opponents will likely be minimal at this early stage of the game. Instead the Player will be focused on learning the movement controls and improving turn timing.

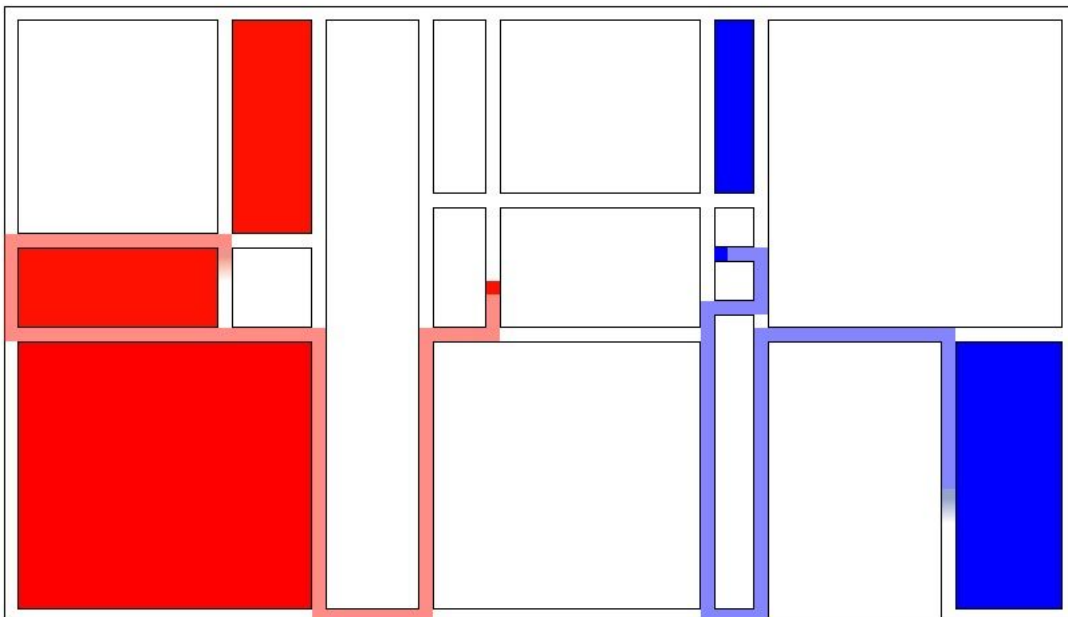
In the sample below, the Player is separated from the blue opponent by a central block that is larger than the Players field of view. Only one path connects the two areas. This sample demonstrates how the odds of early-game conflict can be lessened simply by factoring in requirements like these at the conceptual stage of level design.



Storyboard cont.

As gameplay progresses, the Player learns to improve turn timing and frequency, thereby maximising their average speed and trail length over time. Sustaining a high speed and trail length will cause the field of view to expand, revealing more of the game area to the Player. As new paths and objectives become apparent, risk management and odds of conflict will increase.

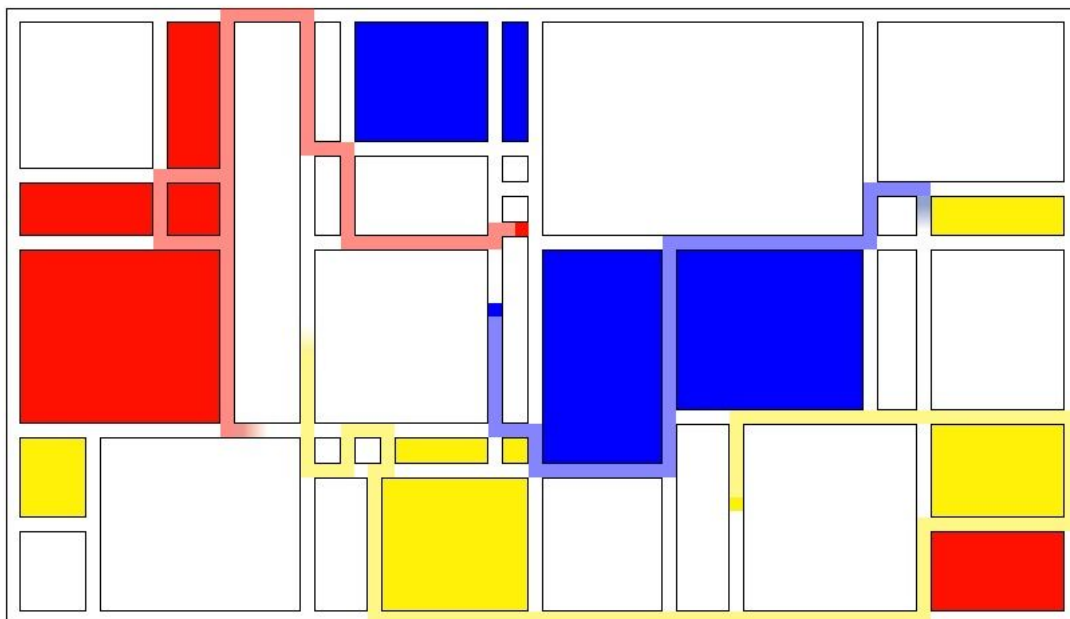
In the sample below, the Player has ventured towards the center of the field of view, bringing them closer to the blue opponent. Proper risk management is now essential as the odds of conflict are about to be drastically affected by the Player's impending decisions. At this stage of the game the Player is encouraged to develop their playstyle by experimenting with both more aggressive and more defensive or passive strategies.



Storyboard cont.

Toward the end of the game the field of view will be expanded, revealing the full game area to the Player. Risk management and territorial conflict will be constant at this stage of the game. When enough territory has been captured the exit will be revealed. The Player must reach the exit in order to achieve the victory condition. Navigating the field of play without crashing or worse is now highly challenging due to the increased speed and trail length of opponents.

In the sample below the Player has made an aggressive move against the blue opponent. This will likely result in a huge advantage for the Player as they are now able to recapture the blue opponent's undefended territory. The yellow opponent still presents a significant threat to the Player.



Reusability

Mechanics

The game mechanics are embedded within the game system and will change dynamically as the Player's skill level increases. As play progresses, use of core mechanics will remain balanced.

Block (Sprite GameObject)

All GameObjects are representative of values within a 2D array.

Moving Objects (Player, Enemy)

All moving objects have the same constraints within the environment. We will write a base class that all moving objects will inherit. This class will constrain moving objects to set positions within the grid array and contain any interaction that occurs between objects.

Using a generic class allows for greater consistency and reusability between moving objects, creating a simplified interface that each moving object will inherit. This allows the programmer to focus on more complex variations, behaviours and input for each object type.

Trail Controller (Player, Enemy)

The *trailController* class is a generic class that will attach itself to every instance of the *movingObjects* class.

The *trailController* class controls trail positions and length for any object that it is attached to.

Logical Models

Object Oriented Models → compilation available [here](#)



Generic

Generic - classes			
MovingObjects(player & enemy)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
currentYDir	int	GetComponent<>()	CheckNextPosition()
currentXDir	int		DoInteraction()
speed	float		CheckCornerTiming()
length	int		IncreasePlayerLength()
maxLength	int		IncreasePlayerSpeed()
maxSpeed	float		DecreasePlayerLength()
playerPosition	int[,]		DecreasePlayerSpeed()
playerNextPosition	int[,]		SetNextPosition()
compiledLevelReference	int[,]		
TrailController (player & enemy)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
parent	GameObject	GetComponent<>()	UpdatePublicArray()
Length	int		CheckIfTrailSurroundsBuilding()
trailPositions	int[,]		NeutralizeBuilding()
parentPosition	int[,]		CaptureBuilding()
levelReference	int[,]		



Player

PLAYER - Object			
PlayerController (player)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
state	String	GetComponent<>()	KeyboardController()
upInput	bool		PassDataToMovingObjectsClass()
downInput	bool		GetDataFromMovingObjectsClass()
rightInput	bool		
leftInput	bool		

Logical Models cont.



Enemies

ENEMY - Object			
EnemyController (enemy)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
state	String	GetComponent<>()	KeyboardController()
hasPath	bool		FindPathToPosition()
nodesInPath	int[]		FindBuilding()
currentNode	int		ConvertCurrentNodeTo2DArray()
currentNodeInPath	int[,]		PassDataToMovingObjectsClass()
currentDirInPath	int[]		GetDataFromMovingObjectsClass()
FindBuilding (enemy)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
lengthRef	int	GetComponent<>()	BuildingChoice()
positionRef	int[,]	Random.Range()	RandomMoveFarAway()
buildingsInRange	GameObject[]		CheckWithinSize()
randomBuilding	GameObject		CheckWithinDistance()
buildingPerimeter	int		CheckOverDistance()
buildingPosition	int[,]		
maxDistance	float		
randomChoice	int		
FindPathToPosition (enemy)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
startPos	int[,]	GetComponent<>()	PlotPositions()
endPos	int[,]		CalculateSurroundingNodesDistance()
distance	float		PickNextNode()
currentNodeDistance	float		CheckPath()
levelReference	int[,]		Convert2dArrayNodesInto1dArray()
gridHeatMap	int[,]		PassArrayToEnemyController()
selectedNodes	int[]		

Logical Models cont.



Event Handlers

EventHandler			
UnlockRegion (EventHandler)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
timer	float	GetComponent<>()	UnlockRegion()
maxTimer	float	Random.Range()	SetRandomTimer()
regionToUnlock	int		
regions	GameObject		
UnlockExitRegion (EventHandler)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
regions	GameObject	GetComponent<>()	WaitForRegionsToBeUnlocked()
isActive	bool		UnlockExit()
regionsActive	int		
regionsUntilActive	int		
CameraHandler (EventHandler)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
State	string	GetComponent<>()	ChangeCameraSize()
playerPos	GameObject	Vector3.Lerp()	FollowPlayer()
cameraPos	GameObject		
playerSize	int		
GenerateEmptyGameObjects (EventHandler)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
gridSizeX	int[,]	GetComponent<>()	GenerateGameobjects()
gridSizeY	int[,]		PushArrayToPublicData()
arrayOf2dGameObjects	GameObject[,]		

Logical Models cont.



Event Handlers cont.

CompileLevel (EventHandler)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
level	sprite[]	GetComponent<>()	Add2dArraysTogeather()
playerPos	int[,]		PushArrayToPublicData()
enemy1Pos	int[,]		
enemy2Pos	int[,]		
playerTrail	int[,]		
enemy1Trail	int[,]		
enemy2Trail	int[,]		
tempCompiledLevel	int[,]		
DrawLevel (EventHandler)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
GameObjectArray	GameObject[]	GetComponent<>()	GetCurrentCompiledLevel()
sprites	Sprite		MinusLastAndCurrentLevels()
compiledLevel	int[,]		LookForChangesIn2dArray()
lastCompiledLevel	int[,]		NoteChangesInArray()
comparedLevelChanges	int[,]		UpdateGameObjectsWithChangedValues()
positionsWithChanges	int[,]		
ScoreHandler (EventHandler)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
levelRef	int[,]	GetComponent<>()	ReadScoreFromLevelArray()
PlayerScore	int		
Enemy1Score	int		
Enemy2Score	int		

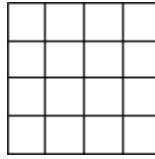
Logical Models cont.



Public Reference Data

Public Reference data			
Level (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
Map	int[,]		
PlayerPos (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
position	int[,]		
Enemy1Pos (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
position	int[,]		
Enemy2Pos (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
position	int[,]		
Compiled2dArray (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
contents	int[,]		
GameObjectArray (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
contents	int[,]		
PlayerTrail (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
positions	int[,]		
Enemy1Trail (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
positions	int[,]		
Enemy2Trail (ref Data)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
positions	int[,]		

Logical Models cont.



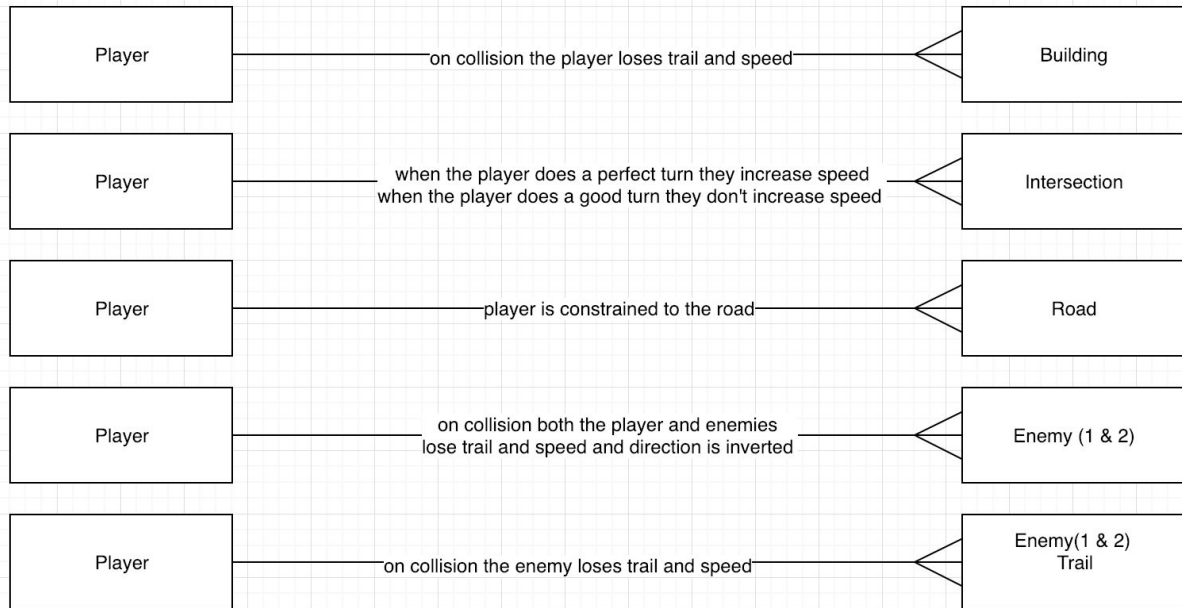
Block

BLOCK - sprite gameobject			
Block (sprite gameobject)			
DATA NAME	DATA TYPE	Unity Function	CUSTOM FUNCTION
ObjectState	string	GetComponent<>()	ChangeSprite()
isRoad	bool		
isBuilding	bool		
isPlayer	bool		
isEnemy	bool		
isTrail	bool		
isHiddenRegion	bool		
isExit	bool		

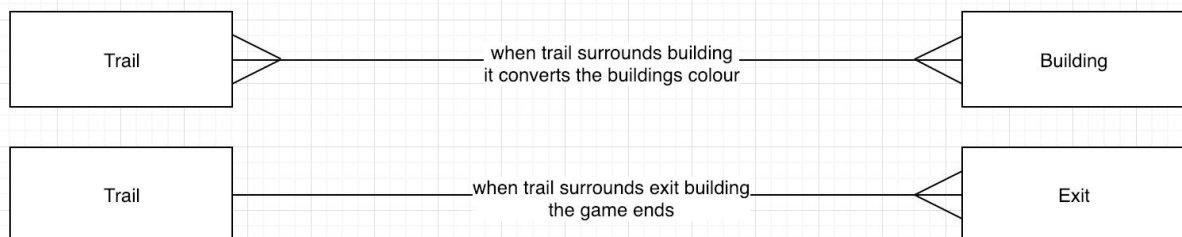
Logical Models cont.

Entity Relationship Diagrams

Player

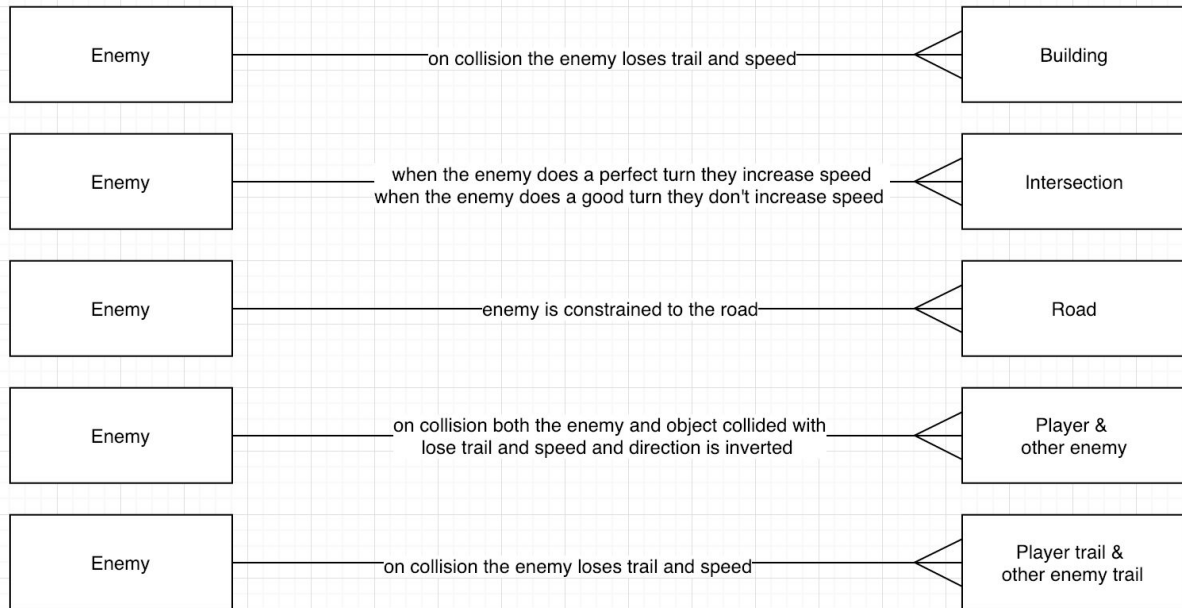


Trail (all)

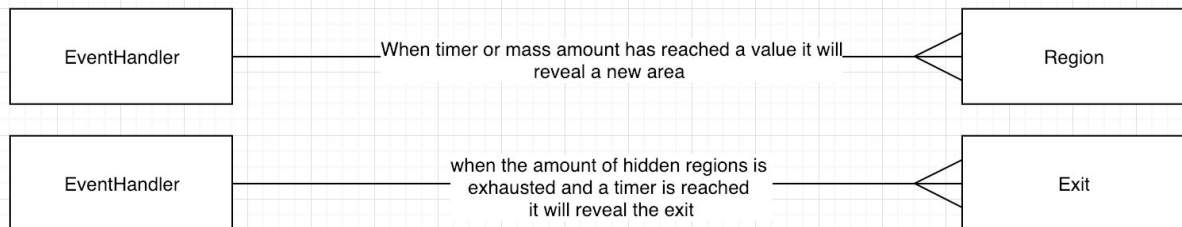


Logical Models cont.

Enemy (1 & 2)

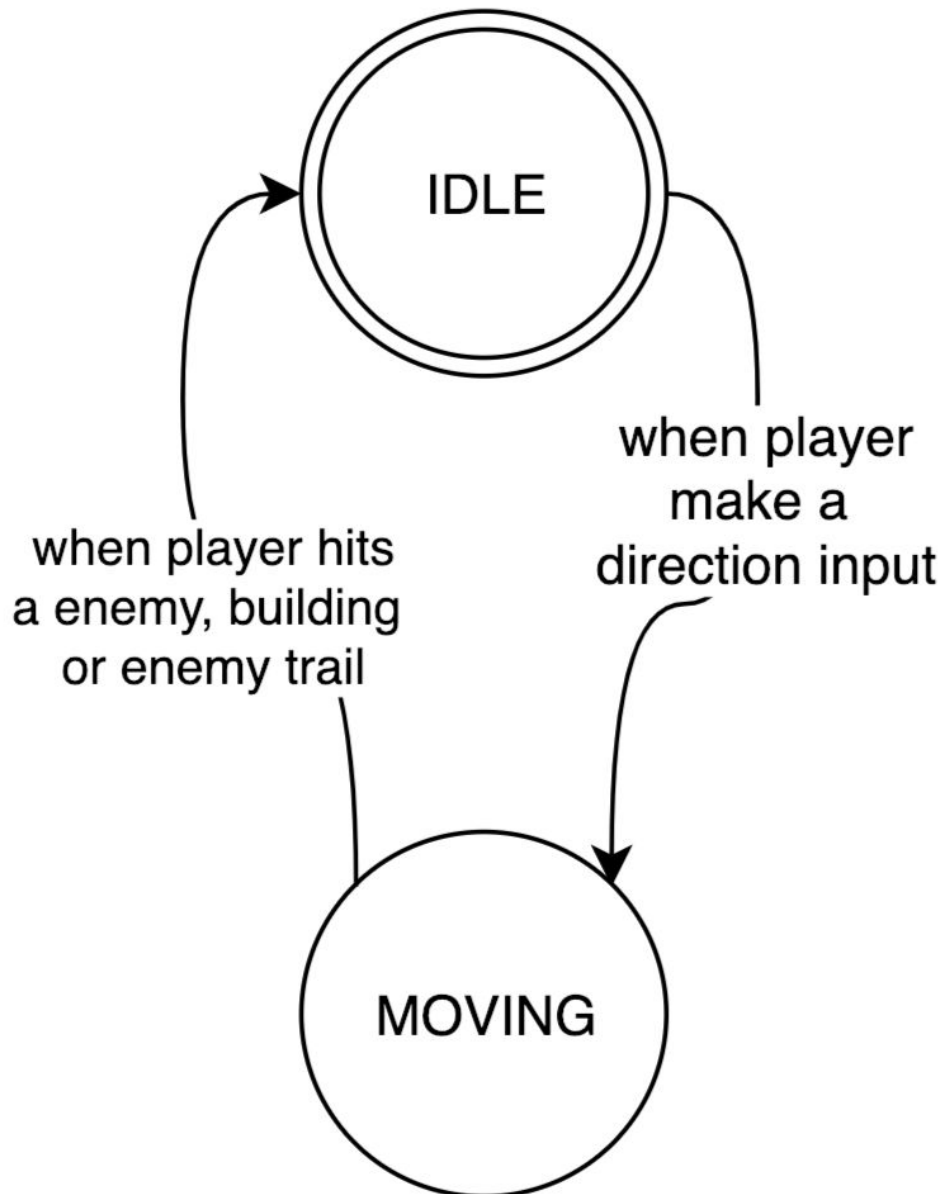


EventHandler

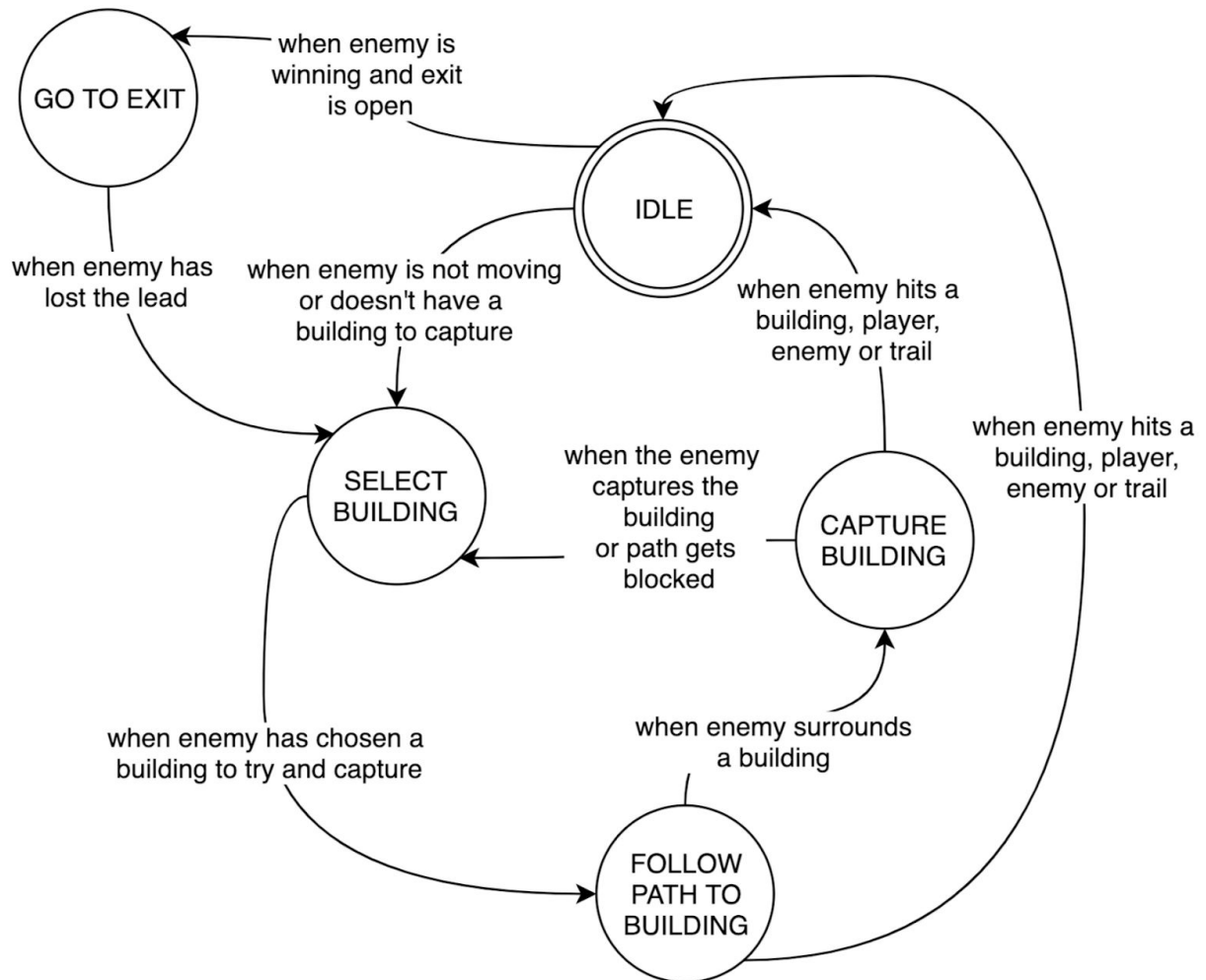


Logical Models cont.

Finite State Machine Player

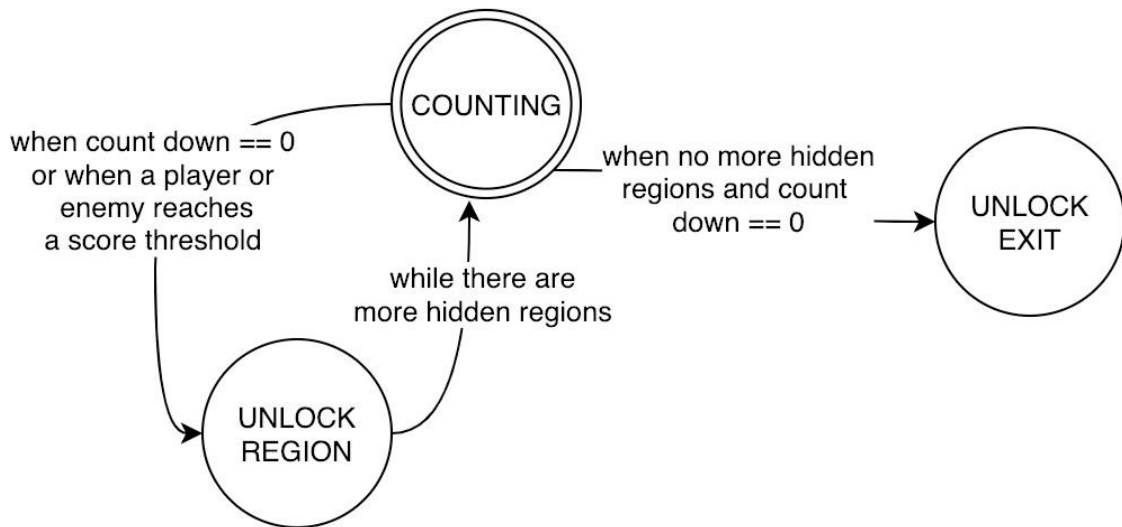


Logical Models cont.

Finite State Machine Enemy (a.k.a. Opponent)

Logical Models cont.

Finite State Machine Event Handler



Finite State Machine Buildings

