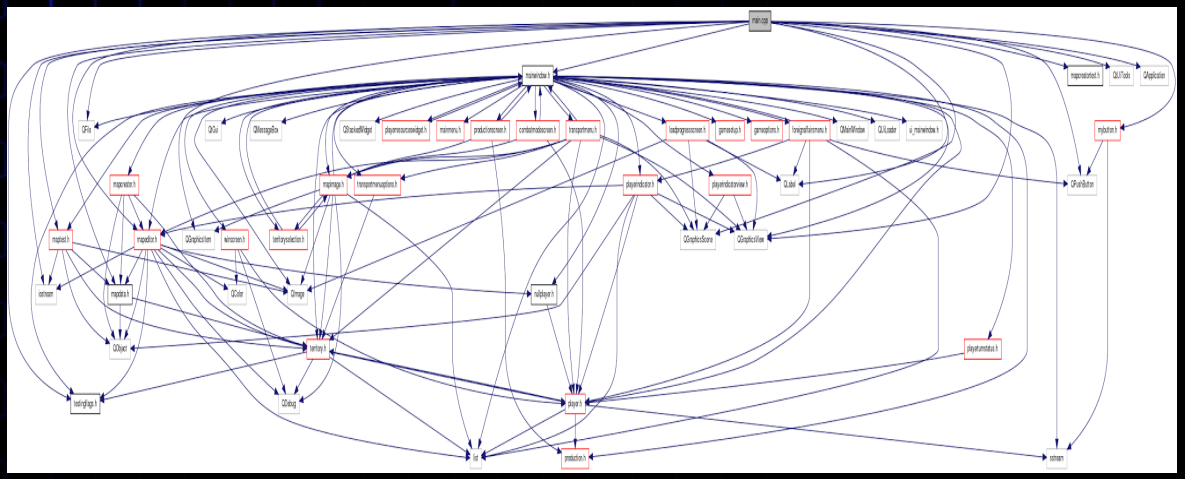
by Transitive Properties

Who We Are

Transitive Properties is composed of...

Andrew "Ringleader" Ring
Luis "Chocobo" Palacios
Yossi ø Katzin
Charles "Magical" Myers

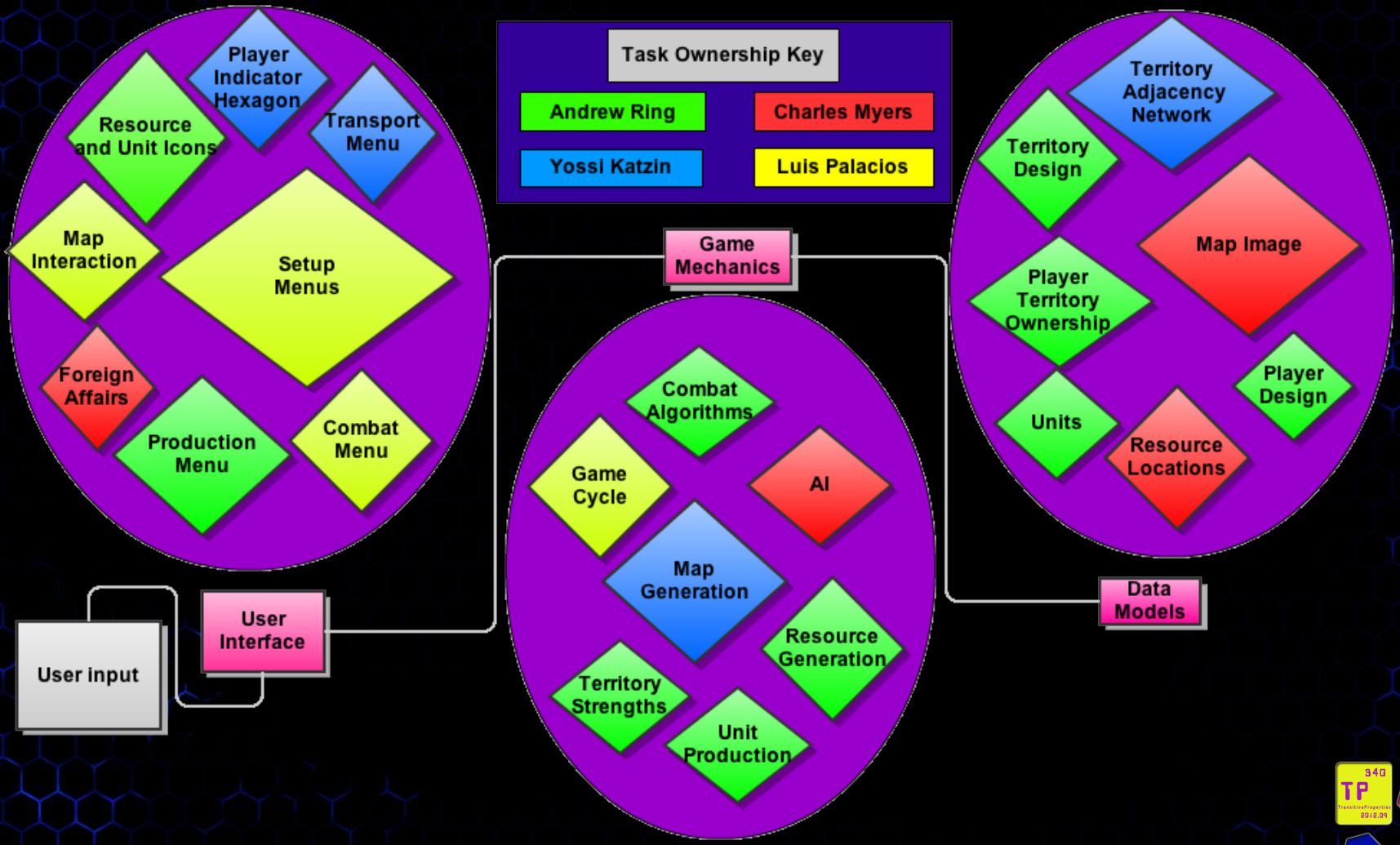
The Real System...



... is too complicated to explain here.

So...

Architecture



Software Design Techniques

- Agile / SCRUM-ish
- Singleton
- Iterator
- Null Object
- Publish/Subscribe
- Model View Controller

Whodunnit?

Andrew	Charles	Luis	Yossi
• Basic Structures • Game Design • Combat Algorithms • Resource Generation and Placement • Territory Selection • Production • Winning • Testing • Scheduling	• Custom Map Drawing • Map Interface • Icon Drawing • Foreign Affairs Menus • Basic AI	• Storyboards • Splash Screen • Main Menu • Options Menu • Combat GUI • Territory Tapping • Hot Seat Multiplayer • Custom Buttons • Load Screen	• Dynamic Map Generation • Player Indicator • Transport

High level review of phases/screens

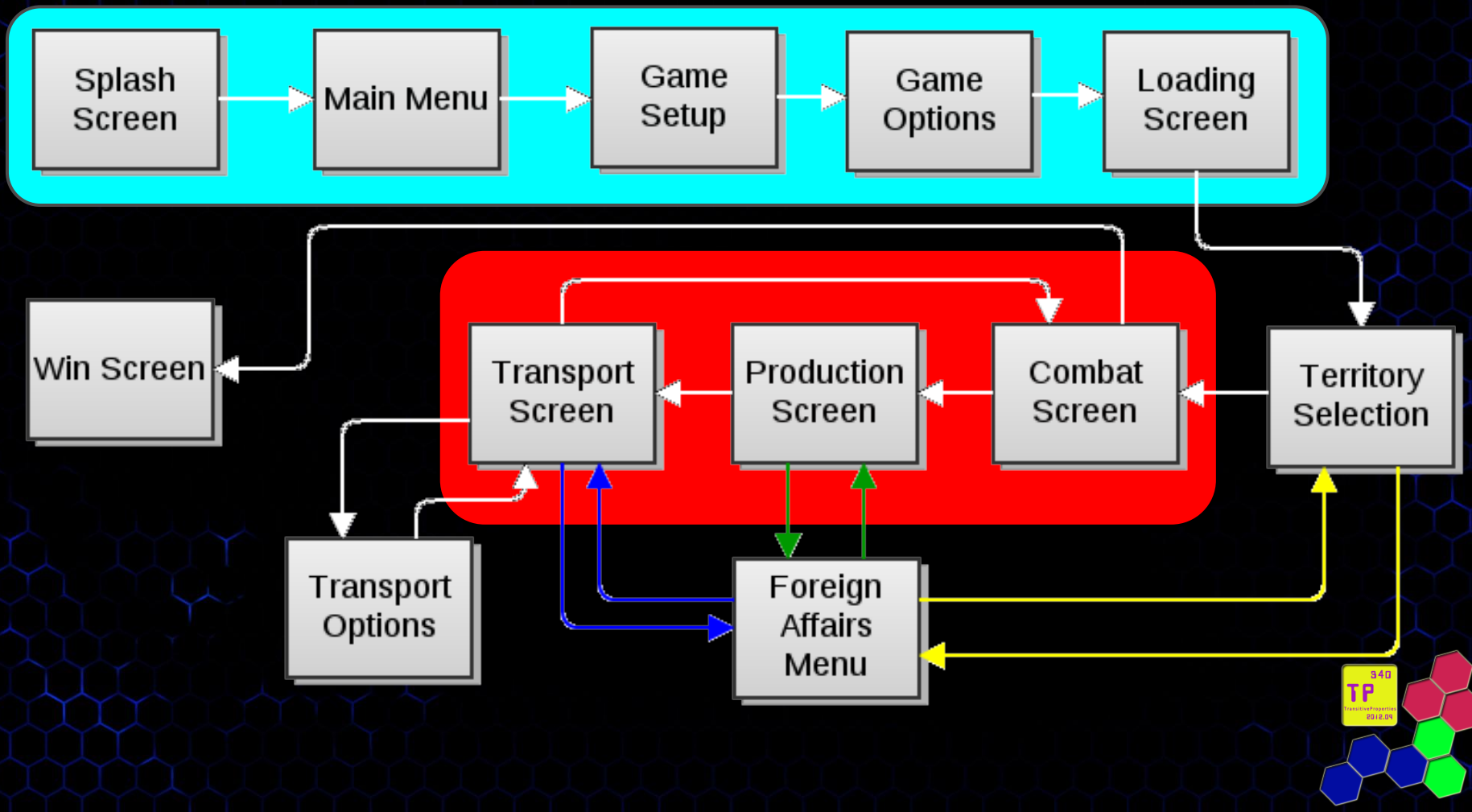
All screens are Widgets created in a hybrid (designer/programmatic) manner to allow support of multiple screen sizes.

All Widgets are maintained within a QStackedWidget within our QMainWindow class.

Game cycle/turn logic is controlled within our QMainWindow class.

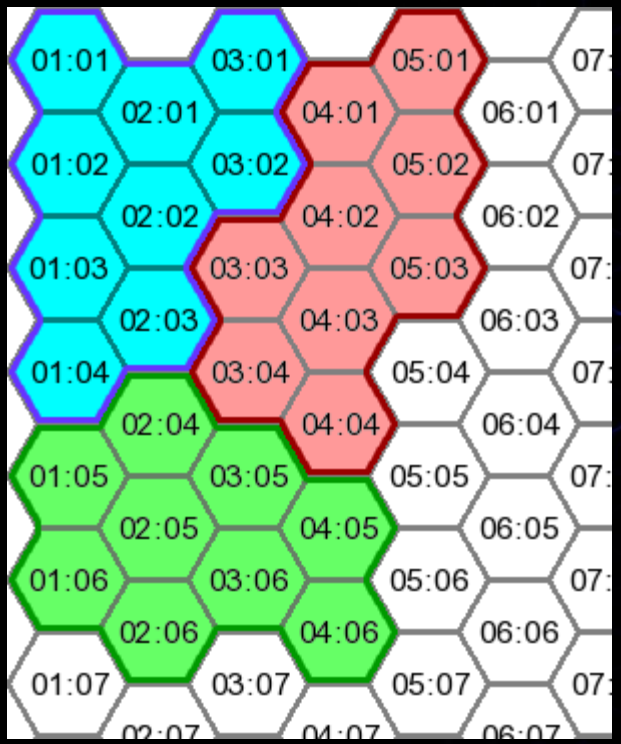
Extensive use of signals/slots made transition between phases/screens seamless.

Screen Flow


Splash Screen
Main Menu
Game Setup
Game Options
Loading Screen
Win Screen
Transport Screen
Production Screen
Combat Screen
Territory Selection
Transport Options
Foreign Affairs Menu

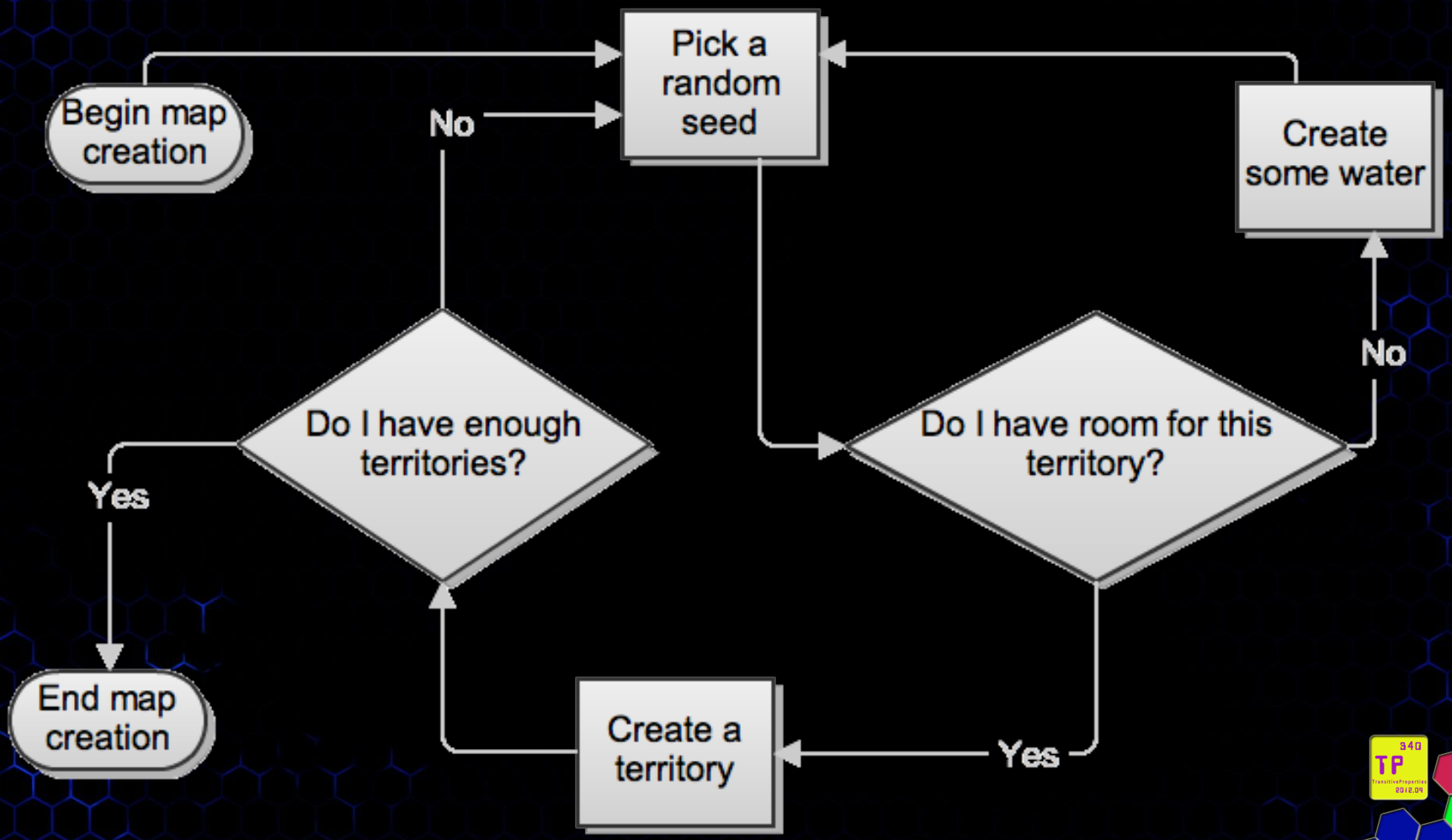
Map Creation (Dynamic)

The map is a two-dimensional array of hexagonal cells, or "hexes".



Map creation consists of grouping hexes for the game's territories. A hex can be in a group with any of its neighbors.

Map Creation algorithm



Map Drawing (with updating)

The map model is an array of "hexes," the units that form a territory.

Each hex is scan-converted individually. Its color is determined by which territory it is part of.

Borders between hexes are drawn if they belong to different territories.

Icons are drawn over the hexes afterwards.

Territory Selection

Players select an increasing number of territories per round, order following an alternating pass through an ordered list.

Example with three Players A, B and C:

A:1 B:1 C:1 C:2 B:2 A:2 A:3 B:3 C:3 C:4 ...

Given a total numbers of territories and players as T and P respectively, each player selects a number of territories of exactly $\lfloor \frac{T}{P} \rfloor$

Players may also elect to have this done for them using the Auto Select feature, which runs the AI selection routine.

AI Territory Selection

Implemented via the standard template library container adaptor priority_queue.

All territories are added to the queue, which sorts them according to its comparator class.

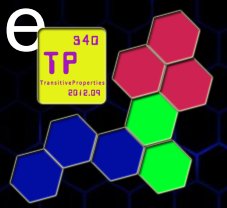
For territory selection, the comparator looks at whether a territory produces resources and its proximity to already claimed territories.

Player Indicator

The player indicator is made of six triangles grouped in a hexagon. For each player, there is a triangle with that player's color. The remaining triangles have the neutral color.

The indicator rotates so that the triangle corresponding to the current player is on top.

If a player's territory is highlighted, his triangle is highlighted and grows.



Foreign Affairs

A programmatically constructed UI that lets a player choose his or her standing towards the other players.

The standings are:

1. Neutral
2. Defend
3. Assist
4. Oppose

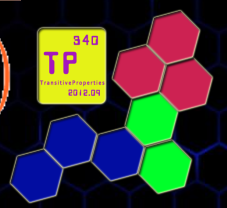
Combat (Interface and Algorithm)

MapImage class extends QGraphicsPixmapItem and has transformations, events overwritten.

Catching TouchEvent() propagated from the MapImage class returns two territories.

These territories are stored for the current player and all the combat occurs at the end of the round.

$$\frac{Player_n + rand(0, VARIANCE)}{\sum_{i=1}^{NumP} Player_i + rand(0, VARIANCE)}$$



AI Combat

Also uses priority_queue.

The comparator class sorts territories based on the relative strengths of attacker and attacked. Territories with low strength ratios are better candidates for attack.

The top two territories in the queue are chosen for the AI player's two attacks.

Production

During the Production phase, players select units to build using their available resources. Cost checking and payment is handled through Resources object operations.

Status changes are controlled through a network of Signals and Slots.

Production is confirmed using a QMessageBox dialogue.

Resources are then generated by all Resource Nodes.

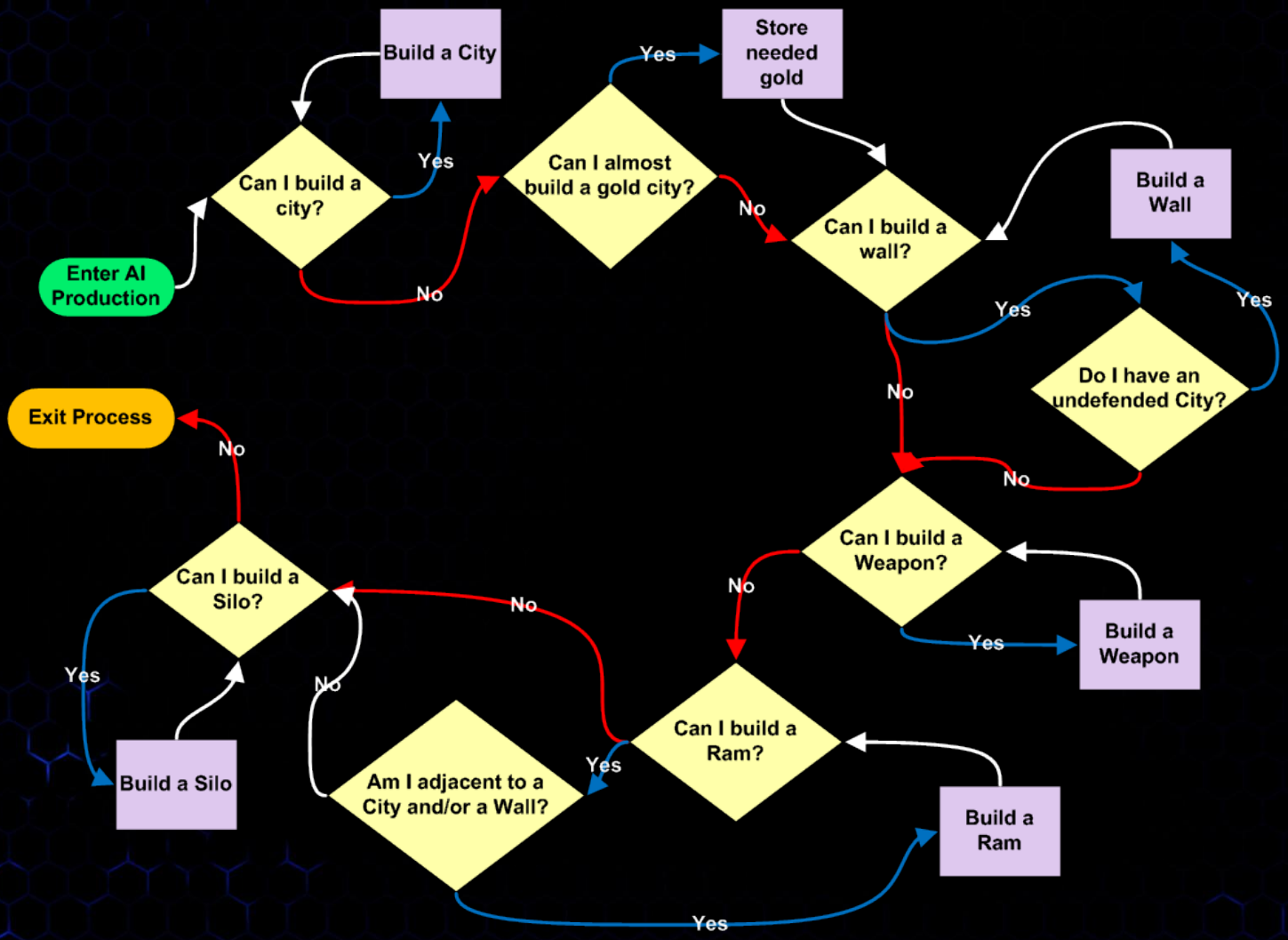
AI Production

Instead of asking the player what to build, the buildChoice() method of that AI player is called repeatedly until it returns a null choice value.

The algorithm considers building each type of unit in a specified order by first looking at the required resources and then at potential locations.

Currently emphasizes a defensive style.

AI Production Decision Flow



Transport

The three implemented movable units are horses, weapons, and rams.

A horse can move a distance of two territories; weapons and rams move one.

A horse can bring a weapon and/or a ram, or neither.

Winning

At the end of each Combat phase, players are checked. If the establish the following conditions:

Player X controls all player owned territories

OR

[*X owns at least the required number of City units* **AND** *more than any other player*]

The Player X wins. The game then transitions to a win screen colored using the winning Player's fill color.

Congratulations!

Transitive Properties is the WINNER!

Questions?