1. Load map
2. Load Monster Queue data
3. Set variables
    a. Load main screen
    b. Initialize scores, heath & items

3. Draw Visable Area
    a. Set window position
    b. Interpret map - is it a wrap around map?
    c. Draw floor and wall tiles
    d. Draw heroes
    e. Draw items
4. Check for input
5. Run Monster Queue
6. Repeat at 3 until next level or death

The map is at $1d00, the last block of memory before the custom characters, and the first custom character is the floor and the second is the wall. The level editor allows these first two characters to be changed with each level. The color information for the walls and floor is tagged along with the monster queue data.

Memory map usage:
$4000-$57ff ML
$1d00-$1dff Map
$1e00-$1f00 Characters
$5800 - $5fff Monsters
$6000 - $63ff Items

Variables
$xxxx Parse Hold
$xxxx Treasure Count

Screen Variables
$xxxx Player 1 Score
$xxxx Player 1 Health
$xxxx Player 1 Items
$xxxx Level
$xxxx Credits
$xxxx Player 2 Score
$xxxx Player 2 Health
$xxxx Player 2 Items

Ghost:	Health	Speed	Xpos	Ypos	Target	
Grunt:	Health	Speed	Xpos	Ypos	Target	
Demon:	Health	Speed	Xpos	Ypos	Target	Status
Lobber:	Health	Speed	Xpos	Ypos	Target	Status
Sorcerer:	Health	Speed	Xpos	Ypos	Target	Status
Death:	Big Health		Xpos	Ypos	Target	
Thief:	Yet to determined if will be included in this version					

The Xpos and Ypos are 0 to 31 and take up the first byte.

Targetting is Free until the monster is live within the visable area.
Targetting attaches to which ever player is closest and isn't changed unless the monster takes damage, then it is set back to Free.
When targetting is set to No Change, the monster will stop moving (this is enabled when all active players are invisible).

Health is the same for all except for Death which use the same byte as status. When Death attacks, his health is diminished until 0 then it is removed.

Status for monsters that shoot contains if they have a shot active - 0 if they don't. This is decremented each cycle, and a value delays the time before shooting again.

Key:	Xpos	Ypos	Quantity		(???xxxxx)(yyyyyqqq)
Potion:	Xpos	Ypos	Type		(???xxxxx)(yyyyyttt)
Food:	Xpos	Ypos	Quantity		(???xxxxx)(?yyyyyqq)
Treasure	Xpos	Ypos	Count		(???xxxxx)(??yyyyyc)
Exit:	Xpos	Ypos	Level		(???xxxxx)(yyyyyLLL)
Door:	Xpos	Ypos	Length	Direction	(xxxxxddm)(yyyyynnn)
Teleport:	Xpos	Ypos	ΔXpos	ΔYpos	(??xxxxxm)(???yyyyy)
Trap:	Xpos	Ypos	Special	...	(??xxxxxm)(???yyyyy)
Bones:	Xpos	Ypos	Health	Rate	(xxxxxhhh)(???yyyyy)(rrrrrrrr)
Generator:	Xpos	Ypos	Health	Rate	(xxxxxhhh)(???yyyyy)(rrrrrrrr)

The quantity in the key is for when a player dies in a two player situation, their keys are left behind next to their bones.

The potion blue and orange potions have no difference. The other 4 potions are the special power ups. One extra item group with the potions is the amulet of invisibility.

Food is set to one of 3 different values.

Treasure always rewards 100 points it exists on a Treasure Room map and instead is counted.

The Exit graphic to display is determined by the offset. The offset determines which level to load next (Current + Offset)

Each animated monster uses 3 bytes:
1 - (xxxxxeee)
2 - (yyyyyhhh/?yyyyytt)
3 - (ssssstt/bbbbbbbb)

Xpos and Ypos use 5 bits each:
They occupy the last 5 bits of the first 2 bytes

Targetting has 2 bits: (tt)
00 - Free
01 - Player 1
10 - Player 2
11 - No change

Health has 3 bits: (hhh)
Big Health uses 8 bits: (bbbbbbbb)

Status:
Entity uses 3 bits (eee)
Shot uses 4 bits (ssss)

Most items use 2 bytes:
1 - (???xxxxx)
2 - (yyyyysss)

Xpos and Ypos use 5 bits each:
They occupy the 5 bits of the first 2 bytes

Each item is interpreted differently:
q - Quantity
t - Type of potion
000 - Cyan
001 - Orange
010 - Move Speed?
011 - Shot Speed
100 - ?
101 - Fight Strength
110 - Invisibility
c - Treasure Room Count
L - Level offset
m - Another segment
n - Length of segment
d - Segment direction

A 2x2 character tile is used. The 15x15 area is first filled with the floor character. Afterwards if a bit is a 1 then a wall character is drawn. Each tile byte is decrypted as follows:

Each byte contains a value 0-15:
bit 0 - NW corner of tile
bit 1 - NE corner of tile
bit 2 - SW corner of tile
bit 3 - SE corner of tile

Determine where the half tile row and half tile column are draw.

1. Clear offset (0x0)
2. Are both VisualAreaXpos and VisualAreaYpos odd?
3. If so, draw half col at col 0 and half row at row 0
    b. set offset to 1x1
4. Is only the VAXpos odd?
5. If so, draw half col at 15 and half row at 0
    b. set offset to 1x0
6. Is only the VAYpos odd?
7. If so, draw half col at 0 and half row at 15
    b. set offset to 0x1
8. If neither are odd, draw half col at 15 and half row at 15

Draw the remaining 14x14 layout starting at one of the following: 0x0; 0x1; 1x0; 1x1

1. If offset is 0x0 draw rest at 0x0
2. If offset is 0x1 draw rest at 0x1
3. If offset is 1x0 draw rest at 1x0
4. If offset is 1x1 draw rest at 1x1

Would it be better to use a 4x1 tile?

Would it be better to only draw new tiles and copy and shift existing visual area?

14 x 14 Draw Method #1

```
lda $visable_area
tax
and #1
beq columncheck
txa
and #16
beq row_odd
both_odd:
row_odd:
columncheck:
txa
and #16
beq  no_odd
column_odd:
ldy #0
no_odd:
draw14x14:
lda ($fc)
sta $fe
and #1
mod1: sta 7768,x
lsr $fe
lda $fe
and #1
mod2: sta 7769,x
lsr $fe
lda $fe
and #1
mod3: sta 7790,x
lsr $fe
lda $fe
and #1
mod4: sta 7791,x
inc $fc
inx
inx
cpx #16
bne draw14x14
rts
```

14 x 14 Draw Method #2

```
lda $visable_area
tax
and #1
beq columncheck
txa
and #16
beq row_odd
both_odd:
row_odd:
columncheck:
txa
and #16
beq  no_odd
column_odd:
ldy #0
no_odd:
draw14x14:
lda $map,y
sta $fc
lda #1
bit $fc
beq n2
mod1: sta 7768,x
n2: lsr $fc
bit $fc
beq n3
mod2: sta 7769,x
n3: lsr $fc
bit $fc
beq n4
mod3: sta 7790,x
n4: lsr $fc
bit $fc
beq n5
mod4: sta 7791,x
n5: iny
inx
inx
cpx #16
bne draw14x14
rts
```