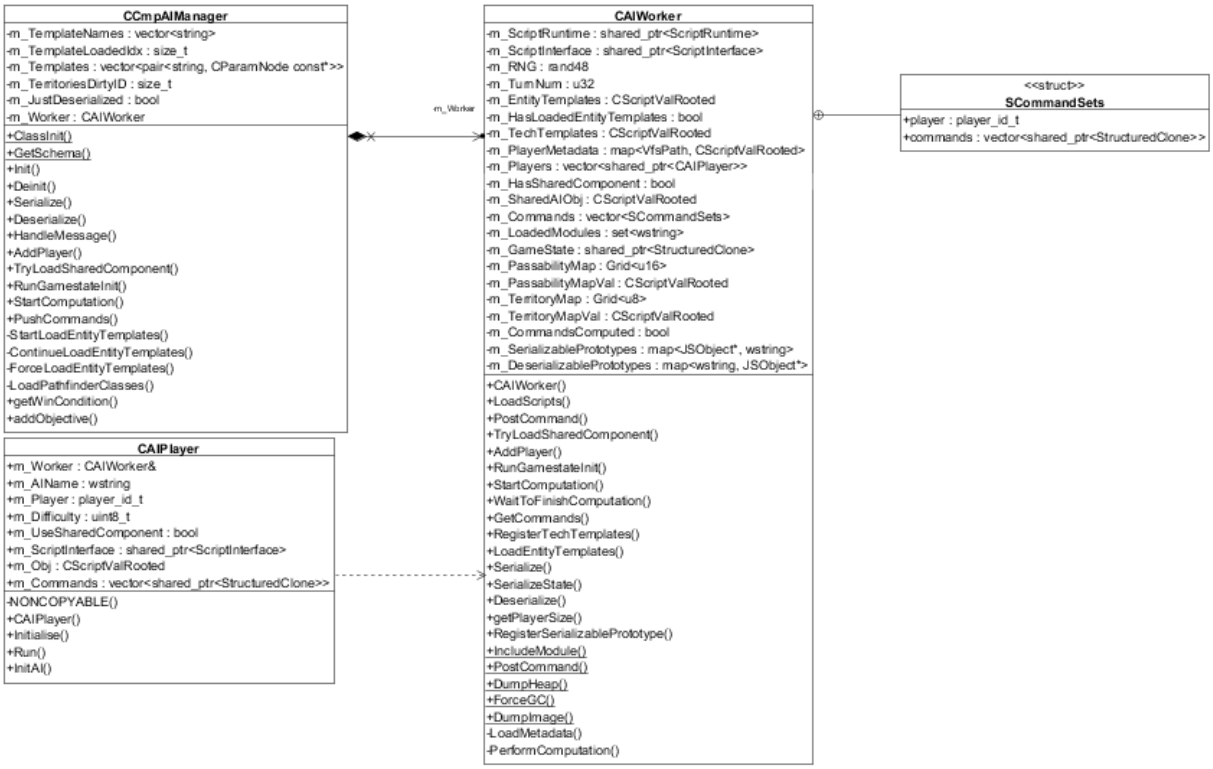
Report for Project 2

New Feature in 0 A.D: Additional feature to display the objectives for a given scenario:-

In 0 A.D, we have various scenarios like you need to defeat the enemy or build the monastery first or conquer all the relics, etc. Once you the start playing the game we do not have anything to suggest the objective of the game to win is. So adding the objective for the game while playing the game will definitely help the player to know the requirements to finish the scenario. The below UML diagram shows the main class and some sub classes in the CCmpAIManager.cpp.



CCmpAIManager.cpp file mainly deals with player AI interface, territory layout and game state.
CCmpAIManager has an association relation with CAIWorker.
CAIWorker deals with the interface related activities of the worker or players in game. This deals with the activities like the players he can attack, work with, civilization he can play, structure of the player, etc. The player Id and commands or operations he can perform is fetched from the structure SCommandSets.
CAIPlayer is deals with the interface related to the user. This is dependent on CAIWorker so there is a dependency relation between these two.

So here I want to implement two new methods “getWinCondition()” and “addObjective()”.
The first method is used to get the winning condition for the game. As for all the scenarios existing in the game the winning scenarios can be defined here and also be fetched based on the gameplay.
The second method is used to display the criteria to win by adding the objectives to the game. So it can fetch the winning criteria for the game and display a GUI for the player while playing the game.
I have included these two methods in CCmpAIManager class.

Change Plan:-

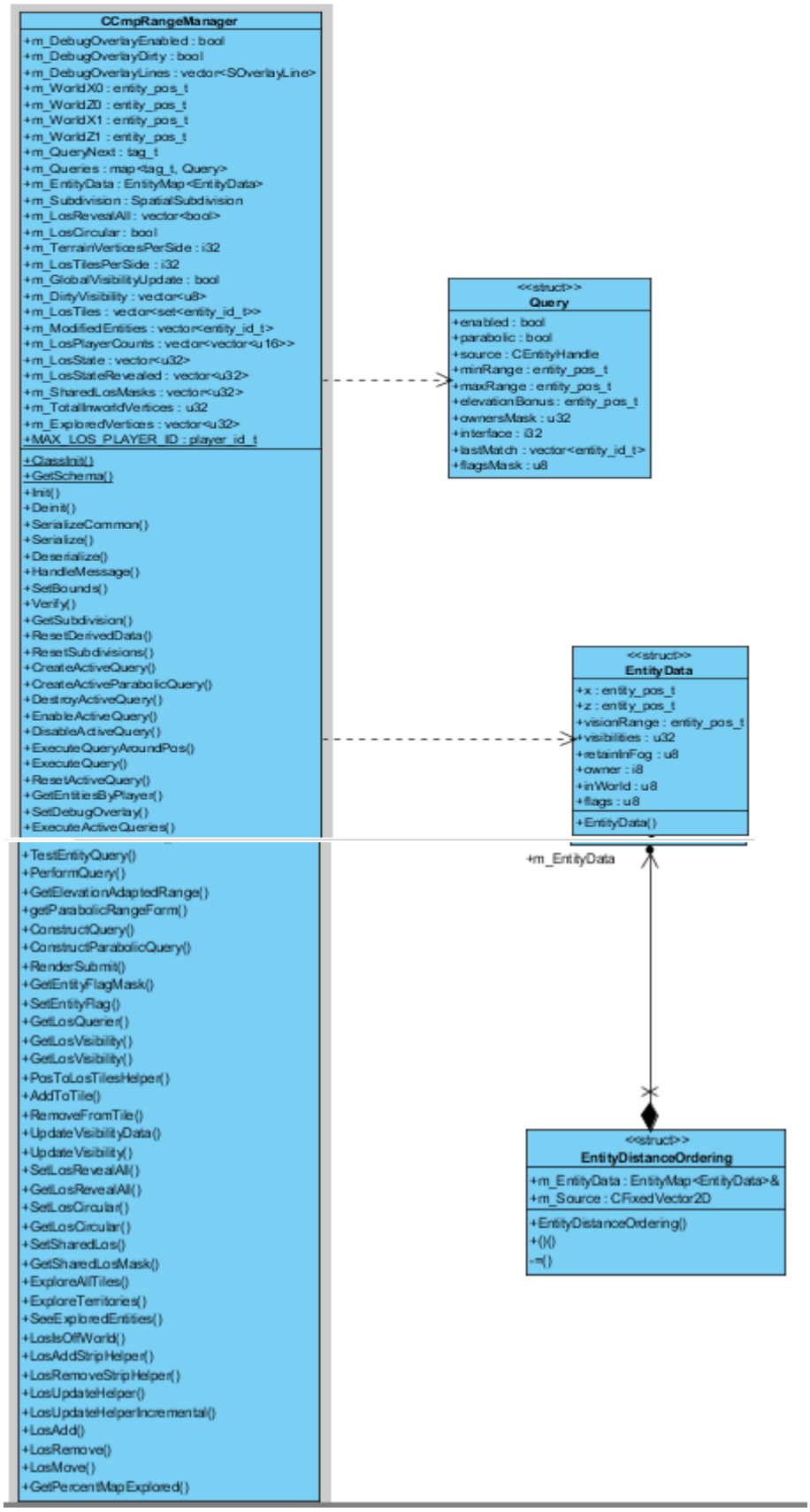
- Define and implement getWinCondition() and addObjective() methods in CCmpAIManager class.
- getWincondition()fetches the required winning condition for the selected scenario.
- Get all the winning objectives for the given scenario.
- Store them in a buffer string based on the given scenarios as an input.
- Return these objectives to the addObjective() method.
- addObjective() will create a GUI to display the Objectives to win the game.
- Deploy the code and then test the functionality.

Improvement in 0 A.D: Refactoring the file CCmpRangeManager.cpp

The CCmpRangeManager.cpp is a large file with close to 2100 lines of code and has 58 methods implemented in it. I have noticed that the CCmpRangeManager.cpp file has the range based queries (Range of unit or building) in the world and Line of sight (LOS) effects (like the fog of war) in a single file, which has made the file too large with so many methods. This is a kind of code smell as the class is large. So we need to refactor the class. Refactoring is like re-structuring the class without changing the external behavior of the class. This way it improves the internal structure of the class.

The below UML diagrams shows the class structure for the file CCmpRangeManager.cpp.
The Class CCmpRangeManager depends on structure Query has this class uses the variables from structure Query so there is a dependency relation. Similarly Structure CCmpRangerManager class has a dependency with EntityData.
The structure EntityDistanceOrdering has a association relation with EntityData has the object is created inside the EntityDistanceOrdering structure.

I have cut the image into two parts then pasted it and then resized it so that the Class diagram can fit in a single page and can been seen clearly.


CCmpRangeManager
+m_DebugOverlayEnabled : bool
+m_DebugOverlayDirty : bool
+m_DebugOverlayLines : vector<SOverlayLine>
+m_WorldX0 : entity_pos_t
+m_WorldZ0 : entity_pos_t
+m_WorldX1 : entity_pos_t
+m_WorldZ1 : entity_pos_t
+m_QueryNext : tag_t
+m_Queries : map<tag_t, Query>
+m_EntityData : EntityMap<EntityData>
+m_Subdivision : SpatialSubdivision
+m_LosRevealAll : vector<bool>
+m_LosCircular : bool
+m_TerrainVerticesPerSide : i32
+m_LosTilesPerSide : i32
+m_GlobalVisibilityUpdate : bool
+m_DirtyVisibility : vector<u8>
+m_LosTiles : vector<set<entity_id_t>>
+m_ModifiedEntities : vector<entity_id_t>
+m_LosPlayerCounts : vector<vector<u16>>
+m_LosState : vector<u32>
+m_LosStateRevealed : vector<u32>
+m_SharedLosMasks : vector<u32>
+m_TotalInworldVertices : u32
+m_ExploredVertices : vector<u32>
+MAX_LOS_PLAYER_ID : player_id_t
+ClassInit()
+GetSchema()
+Init()
+Deinit()
+SerializeCommon()
+Serialize()
+Deserialize()
+HandleMessage()
+SetBounds()
+Verify()
+GetSubdivision()
+ResetDerivedData()
+ResetSubdivisions()
+CreateActiveQuery()
+CreateActiveParabolicQuery()
+DestroyActiveQuery()
+EnableActiveQuery()
+DisableActiveQuery()
+ExecuteQueryAroundPos()
+ExecuteQuery()
+ResetActiveQuery()
+GetEntitiesByPlayer()
+SetDebugOverlay()
+ExecuteActiveQueries()
+TestEntityQuery()
+PerformQuery()
+GetElevationAdaptedRange()
+getParabolicRangeForm()
+ConstructQuery()
+ConstructParabolicQuery()
+RenderSubmit()
+GetEntityFlagMask()
+SetEntityFlag()
+GetLosQuerier()
+GetLosVisibility()
+GetLosVisibility()
+PosToLosTilesHelper()
+AddToTile()
+RemoveFromTile()
+UpdateVisibilityData()
+UpdateVisibility()
+SetLosRevealAll()
+GetLosRevealAll()
+SetLosCircular()
+GetLosCircular()
+SetSharedLos()
+GetSharedLosMask()
+ExploreAllTiles()
+ExploreTerritories()
+SeeExploredEntities()
+LosIsOffWorld()
+LosAddStripHelper()
+LosRemoveStripHelper()
+LosUpdateHelper()
+LosUpdateHelperIncremental()
+LosAdd()
+LosRemove()
+LosMove()
+GetPercentMapExplored()
<<struct>>
Query
+enabled : bool
+parabolic : bool
+source : CEntityHandle
+minRange : entity_pos_t
+maxRange : entity_pos_t
+elevationBonus : entity_pos_t
+ownersMask : u32
+interface : i32
+lastMatch : vector<entity_id_t>
+flagsMask : u8
<<struct>>
EntityData
+x : entity_pos_t
+z : entity_pos_t
+visionRange : entity_pos_t
+visibilities : u32
+retainInFog : u8
+owner : i8
+inWorld : u8
+flags : u8
+EntityData()
+m_EntityData
<<struct>>
EntityDistanceOrdering
+m_EntityData : EntityMap<EntityData>&
+m_Source : CFixedVector2D
+EntityDistanceOrdering()
+()()
-m()

To make the code cleaner and understandable the classes has to be separated out into two separate .cpp files. That is place all the methods related to Line of Sight inside a new file. In this way we can refactor the class.

Change plan:-

- Create a new .cpp file CCmpRangeManager_los.cpp file.
- Separate all the methods related to Line of sight (LOS) into the CCmpRangeManager_los.cpp file. The below methods are related to LOS. GetLosQuerier(),GetLosVisibility(),AddToTile(),RemoveFromTile(),UpdateVisibilityData(), UpdateVisibility(),SetLosRevealAll(),GetLosRevealAll(),SetLosCircular(),GetLosCircular(),SetSharedLos(),GetSharedLosMask(),ExploreAllTiles(),ExploreTerritories(),SeeExploredEntities(),LosIsOffWorld(),LosAddStripHelper(),LosRemoveStripHelper(),LosUpdateHelper(), LosUpdateHelperIncremental(),LosAdd(),LosRemove(),LosMove(),GetPercentMapExplored()
- Separate the variables related to the methods accordingly.
- This way the size of the file reduces and can easily be extended in future.
- Deploy the code with the changes.

Improvement in 0 A.D: Refactoring the Normal Game Speed

functions_utility.js tracks all the utility function like color of objects, converting the time to milliseconds, game speed and map sizes. In this method we have a method initGameSpeeds() which is used to set the game speed. This method allows the user to select the game speed after starting the game. The default speed is always set to 1 as the game starts.

So instead of hardcoding the default value for the game speed to 1, we can take the value from the user. So we define a variable GAME_SPEED in gamesetup.js file where the entire configuration related to the game is stored. So before deploying the game, it can be changed as per the user convenience.

The code snippet for the initGameSpeeds() method in the file functions_utility.js is shown below:

```
function initGameSpeeds() {
     var gameSpeeds = {
          "names" : [],
          "speeds" : [],
          "default" : 0
     };

     var data = parseJSONFromDataFile("game_speeds.json");
     if (!data || !data.Speeds)
```

```
          error("Failed to parse game speeds in game_speeds.json
(check for valid JSON data)");
     else {
          translateObjectKeys(data, [ "Name" ]);
          for ( var i = 0; i < data.Speeds.length; ++i) {
               gameSpeeds.names.push(data.Speeds[i].Name);
               gameSpeeds.speeds.push(data.Speeds[i].Speed);
               if (data.Speeds[i].Default){
                    gameSpeeds["default"] = i;
               data.Speeds[i].Speed = 1.0;
               data.Speeds[i].Name = "Default";
               }
          }
     }
     return gameSpeeds;
}
```

We declare the variable GAME_SPEED in gamesetup.js

```
const GAME_SPEED = 1.3 // time is seconds
```

Code Snippet after changing the hardcoded value

```
     function initGameSpeeds(GAME_SPEED) {
     var gameSpeeds = {
          "names" : [],
          "speeds" : [],
          "default" : 0
     };

     var data = parseJSONFromDataFile("game_speeds.json");
     if (!data || !data.Speeds)
          error("Failed to parse game speeds in game_speeds.json
(check for valid JSON data)");
     else {
          translateObjectKeys(data, [ "Name" ]);
          for ( var i = 0; i < data.Speeds.length; ++i) {
               gameSpeeds.names.push(data.Speeds[i].Name);
               gameSpeeds.speeds.push(data.Speeds[i].Speed);

               if (data.Speeds[i].Default){
                    gameSpeeds["default"] = i;
               data.Speeds[i].Speed = GAME_SPEED;
               data.Speeds[i].Name = "Default";
          }
     }     }
```

```
    return gameSpeeds;
}
```

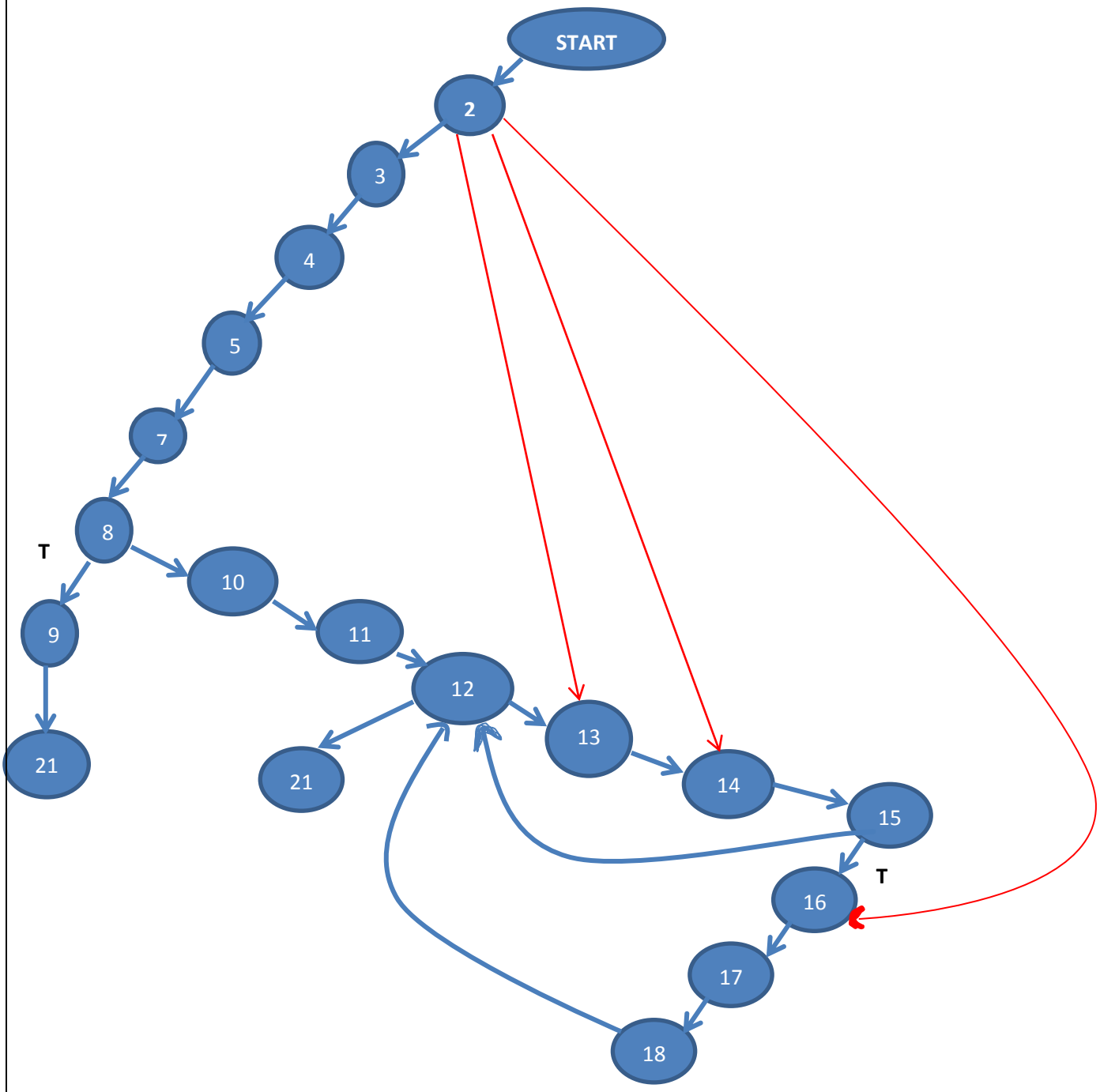
Change Plan:-

- In gamesetup.js file add a variable GAME_SPEED.
- In gamesetup.js file pass variable GAME_SPEED as a parameter to the function initGameSpeeds().
- In functions_utility.js file replace the function declaration by passing a value GAME_SPEED.
- In functions_utility.js file replace the hardcoded speed value with GAME_SPEED variable.
- Change the GAME_SPEED variable to a desired value.
- Deploy the code with the changes.

Considering the below code, PDG is drawn.

```
1.   function initGameSpeeds(GAME_SPEED) {
2.   var gameSpeeds = {
3.   "names" : [],
4.   "speeds" : [],
5.   "default" : 0
6.   };
7.   var data = parseJSONFromDataFile("game_speeds.json");
8.   if (!data || !data.Speeds)
9.   error("Failed to parse game speeds in game_speeds.json
     (check for valid JSON data)");
10.  else {
11.  translateObjectKeys(data, [ "Name" ]);
12.  for ( var i = 0; i < data.Speeds.length; ++i) {
13.  gameSpeeds.names.push(data.Speeds[i].Name);
14.  gameSpeeds.speeds.push(data.Speeds[i].Speed);
15.  if (data.Speeds[i].Default){
16.  gameSpeeds["default"] = i;
17.  data.Speeds[i].Speed = GAME_SPEED;
18.  data.Speeds[i].Name = "Default";
19.  }
20.  }
21.  return gameSpeeds;
22.  }
23.  }
```

Since there is only one variable change in the code there will be no change in the flow. So the PDG remains the same before and after the change. A Program Dependency Graph is combination of Data flow diagram and control flow graph. In a control flow graph nodes are the statements and the edges are the dependencies between the statements.



In the above PDG all the statements are represented by nodes and the edges as dependencies between statements. The control flow is from 2 to 7 and with the statement 8 there is an if condition so the control goes to 9 or 10 based on the if statement data logic. Since we have a for loop condition at statement 12 the statements 15 and 18 are looped to it. Also, there is an if condition at statement 15 so the flow can be 15->16->17->18 or 15->12.

The red lines show the data dependency for the variable gameSpeeds.