Software Design Document (SDD) Template

Software design is a process by which the software requirements are translated into a representation of software components, interfaces, and data necessary for the implementation phase. The SDD shows how the software system will be structured to satisfy the requirements. It is the primary reference for code development and, therefore, it must contain all the information required by a programmer to write code. The SDD is performed in two stages. The first is a preliminary design in which the overall system architecture and data architecture is defined. In the second stage, i.e. the detailed design stage, more detailed data structures are defined and algorithms are developed for the defined architecture.

This template is an annotated outline for a software design document adapted from the IEEE Recommended Practice for Software Design Descriptions. The IEEE Recommended Practice for Software Design Descriptions have been reduced in order to simplify this assignment while still retaining the main components and providing a general idea of a project definition report. For your own information, please refer to [I EE E St d 1016­ 1998](http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieee-SDD.pdf) 1 for the full IEEE Recommended Practice for Software Design Descriptions.

1 http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieee­SDD.pdf

TLD(traffic light decider)

Software Design Document

Name (s): Igor krol, Denis Shapira

Lab Section:

Workstation:

Date: (31/3/2020)

**TABLE OF CONTENTS**

**1. INTRODUCTION 2**

1.1 Purpose 2

1.2 Scope 2

1.3 Overview 2

1.4 Reference Material 2

1.5 Definitions and Acronyms 2

**2. SYSTEM OVERVIEW 2**

**3. SYSTEM ARCHITECTURE 2**

3.1 Architectural Design 2

3.2 Decomposition Description 3

3.3 Design Rationale 3

**4. DATA DESIGN 3**

4.1 Data Description 3

4.2 Data Dictionary 3

**5. COMPONENT DESIGN 3**

**6. HUMAN INTERFACE DESIGN 4**

6.1 Overview of User Interface 4

6.2 Screen Images 4

6.3 Screen Objects and Actions 4

**7. REQUIREMENTS MATRIX 4**

**1. INTRODUCTION**

**1.1 Purpose**

This document contains the complete design description of the *TLD*. This includes the architectural features of the system down through details of what operations each code module will perform and the database layout. It also shows how the use cases detailed in the SRS will be implemented in the system using this design.

The primary audiences of this document are the software developers.

**1.2 Scope**

There are three parts of this system:

* The *data collection and adaptation* gathers information from traffic and adapts it to the algorithm.
* The *algorithm(TLD)* based on the data decides on the lights on each traffic-light for a period of time(cycle).
* The *results adaptation* displays the cycle on the screen with the data from the algorithm.

**1.3 Overview**

Provide an overview of this document and its organization.

**1.4 Reference Material**

*This section is optional.*

List any documents, if any, which were used as sources of information for the test plan.

**1.5 Definitions and Acronyms**

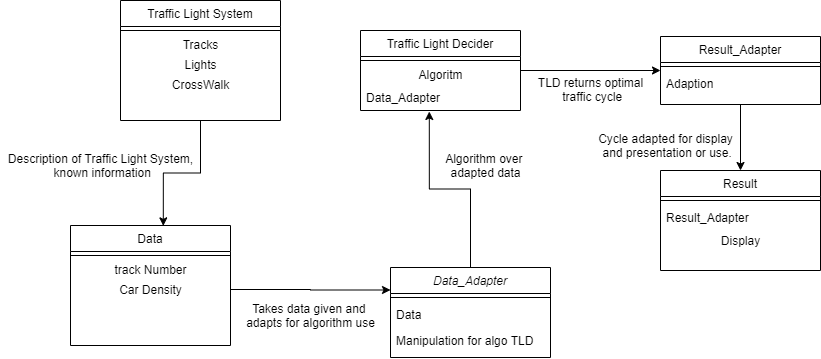
|  |  |
| --- | --- |
| Term | Definition |
| TLD | Traffic light decider |
| Cycle | Results over a period of time |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**2. SYSTEM OVERVIEW**

Give a general description of the functionality, context and design of your project. Provide any background information if necessary.

**3. SYSTEM ARCHITECTURE**

**3.1 Architectural Design**



**3.2 Decomposition Description**

Traffic light system: data which describes the module of the traffic with information

Which doesn’t change.

Data: the information that the system will gather to decide the next cycle.

Data\_Adapter: manipulates the data from Data to better suite TLD.

TLD: the algorithm that calculates the next cycle.

Result\_adapter: manipulates the data from TLD to better suite Result.

Result: displays the information it receives on the screen.

**3.3 Design Rationale**

When deciding on the architectural design we pondered in which way we wanted the software to be divided, After deciding on 3 part division we tried to think on the implementation of each part and realized that we needed to adapt and manipulate the data from each part so we could improve a part without changing the whole software. Or if someone had a better algorithm for example he could switch that part and he will have to change only the adapters.

**4. DATA DESIGN**

**4.1 Data Description**

The information about the roads is saved in the appropriate GameObjects.  
after which the system creates new information based on the details provided and sends the info to the algorithm.

**4.2 Data Dictionary**

Traffic light system:

attribute:

GameObject physical map

GameObject roads

GameObject traffic lights

Data:

attribute:

string name

int density

GameObject points

methods:

onCollisionEnter(collision other)

createCars()

newCPM(string cpm)

Data adapter:

attribute:

List<string> constraintList

int numOfRoads

float timeForCycle

List<int> densityList

methods:

restart()

changeRoad(int name, float cpm)

AlgoRun(int \_nor, List<int> \_density)

changeCycleTimer(string text)

TLD:

attribute:

Junction junction

List<string> constraints

methods:

simulatedAnnealing(Junction start, float t, float end\_t, int limit)

ResultAdapter:

methods:

CycleSegmentsCompute(Junction \_jn)

Results:

attribute:

Junction schedule

methods:

lights(Junction res, float totalTime)

**5. COMPONENT DESIGN**

onCollisionEnter:

if collision with another road

add to constraint list

createCars:

create a car and initialize at the starting location of the road

newCPM:

change the road's density

restart:

remake the densityList

run AlgoRun

changeRoad:

go through the densityList untill you get the name

change the list to the new density

AlgoRun:

recalculate the new total time for a cycle

make a new junction with the parameters

compute the junction

send results to result adapter

ChangeCycleTimer:

change timeForCycle

compute the total time for a cycle

SimulatedAnnealing:

Let s = s0

For k = 0 through kmax (exclusive):

T ← temperature( (k+1)/kmax )

Pick a random neighbour, snew ← neighbour(s)

If P(E(s), E(snew), T) ≥ random(0, 1):

s ← snew

Output: the final state s

CycleSegmentsCompute:

float[] denSum

float[] percent

sum

for i=0 through number of segments

for j=0 through number of the roads

if traffic lights is open

denSum[i]+=junction.getDensity[j]

for each n in denSum

sum+=n

for i=0 through number of segments

if sum!=0

percent[i]=denSum[i]/sum

return percent

lights:

for i=0 through number of segments

for j=0 through numbers of trafficlights

turn on trafficlight named j

wait 2 seconds

for j=0 through nu,bers of trafficlights

turn off trafficlight name j if schedule[j,i]

wait percent[i]\*totalTime seconds

run restart

**6. HUMAN INTERFACE DESIGN**

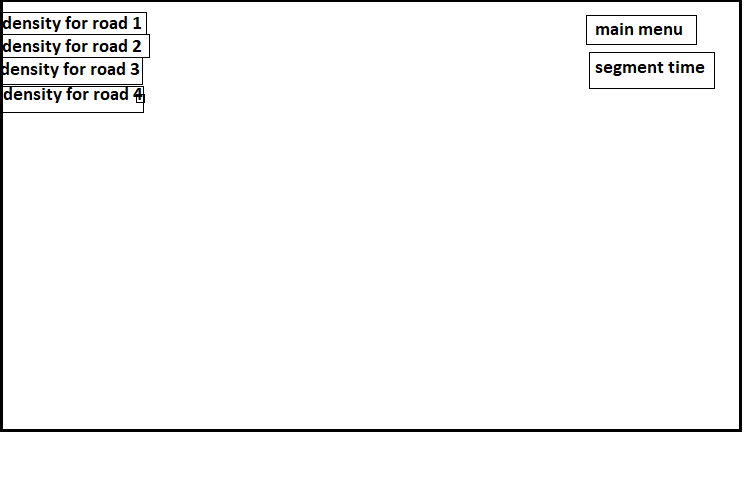
**6.1 Overview of User Interface**

The user can change the road he can currently see from the main menu.

The user can change each road's density.

the user can change how much time each segment in the cycle lasts.

**6.2 Screen Images**



**6.3 Screen Objects and Actions**

Density for road x:

Change the density of a particular road.

Segment time: how much time on average each segment has.

Main menu: go back to the main menu and there you can type which road you want to run the program on.

**7. REQUIREMENTS MATRIX**

Provide a cross­reference that traces components and data structures to the requirements in your

SRS document.

Use a tabular format to show which system components satisfy each of the functional requirements from the SRS. Refer to the functional requirements by the numbers/codes that you gave them in the SRS.

## 3.1 External Interfaces

**3.1.1 Softwere Interface**

Traffic light system

No other requirements

## 3.2 Functions

**3.2.1 Creation**

A junction can be built easily with unity(prefabs) with the data given

**3.2.2 Data collection**

UI input Fields

**3.2.3 Data combination**

Data adapter

**3.2.4 Comibation ranking**

TLD

**3.2.5 Traffic cycle manager**

Result

**3.2.6 Traffic rules Check**

TLD

## 

**3.2.7** **Simulation**

Result

## 3.5 Software System Attributes

The requirements in this section specify the required reliability, availability, security and maintainability of the software system.

### 3.6.1 Reliability

TLD

### 3.6.2 Availability

Unity

### 3.6.3 Security

None (system is offline)

### 3.6.4 Maintainability

Scripts and data is easily changeable

### 3.6.5 Portability

Unity