Software Design Document

Author(s): Brian Bianchi, Andrew Brevick, Cuong Ha

Team #21

University of Minnesota - Twin Cities - Spring 2023

Date: 03/03/2023

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

This software design document describes the architecture and system design of a system that tabulates election ballots and produces election results following the rules of two types of election: Instant Runoff (IR) and Closed List Party (CPL). It will give details of the system design, including the architecture of the object oriented programming (OOP) classes that will compose the system, details of these classes, and data storage and flow. It also explains the features of the system, the primary modes of user interaction, and proposed formats for the user interface and outputs in high-level details, allowing the readers to understand how the components of this system work and interact with each other more easily. Finally, it will show how the system design meets the requirements given in the software requirements specification (SRS) document.

As such, the primary audience for this document is the development team and QA team. The development team will use it to guide the implementation of the system described here. The QA team will use it to ensure testability and also to ensure that proper test cases are written.

1.2. Scope

- This system is an electronic tool to tabulate ballots and to determine the winner(s) in a few common types of election.
- A CSV file containing the election results will be given to the system by the user.
- The system reads and uses the provided data to tabulate and finalize the elections.
- The methodology of how the calculations are done depends on the chosen election type.
- The process is automated to substantially reduce the potential for user error and to decrease the amount of time required for tabulation.

1.3. Overview

The following sections of this document will focus on describing the architectural design of the Voting Tabulating System, as well as the interactions between its high-level components. There will be several diagrams being represented throughout the document, from the most general to more in-depth, along with detailed descriptions to help the reader comprehend the flow of this system more easily.

Section 2 includes an overview of the nominal operation flow of this system, and the election algorithms that will be implemented. Section 3 includes detailed diagrams of the system architecture in the format of UML class diagrams, as well as details of the IR and CPL diagrams in the form of process and sequence diagrams, respectively . Section 4 focuses on details of the data storage and processing for ballot information. Section 5 expands upon the classes presented in Section 3 in more detail, with function descriptions of class methods. Section 6 shows details of the planned user interface. Finally, Section 7 shows how the system design meets the system requirements via a requirements matrix.

1.4. Reference Materials

- Software Requirements Specification
  - Title: Software Requirements Specification for Voting Tabulation System
  - Author(s): Andrew Brevick, Brian Bianchi, Cuong Ha
  - Organization: University of Minnesota - Twin Cities
  - Publication: Spring 2023
  - Source: https://github.umn.edu/umn-csci-5801-01-S23/repo-Team21/blob/main/SRS/SRS_Team21.pdf
- Main Project Description
  - Title: Project 1 - Waterfall Methodology
  - Author(s): Shana Watters
  - Organization: University of Minnesota - Twin Cities
  - Publication: Spring 2023
  - Source: https://canvas.umn.edu/courses/355321/pages/updated-waterfall-election-system-description-with-fixed-cpl-file-formatting?module_item_id=9728200
- SDD Project Description
  - Title: Project 1 - Waterfall Methodology
  - Author(s): Shana Watters
  - Organization: University of Minnesota - Twin Cities
  - Publication: Spring 2023
  - Source: https://canvas.umn.edu/courses/355321/assignments/3090481
- Learn more about Google Style Guide here
  - Source: https://google.github.io/styleguide/cppguide.html
- Learn more about UML here
  - Source: https://www.uml-diagrams.org/
- SDD Template can be downloaded from here
  - Title: Software Design System
  - Publisher: IEEE
  - Publication: 1998
  - Source: https://canvas.umn.edu/courses/355321/assignments/3090481

1.5. Definitions and Acronyms

- IR: Instant Runoff Voting Method
- CPL: Closed Party List Voting Method
- Election officials: authorized people in charge of initiating the system and finalizing the results
- Tie-breaker: method(s) to determine the winner between two or more candidates who have the same amount of votes

2. SYSTEM OVERVIEW

This system is a tool used by the election officials to tabulate ballots, finalize the results, and determine the winner(s) following the rules of Instant Runoff method and Closed Party List method.

The standard operating flow of the system is as follows:

1. **User inputs** - User enters required information including file path to ballot file and election date
2. **Header import** - System parses the ballot file header for information including election type, candidate count and names, and ballot count. Information is verified by the user.
3. **Ballot processing** - Ballots are read into the system and tabulated according to the rules of a particular election type. See the following subsections for these details.
4. **Results generation** - The election winners and other results information is displayed to the user.
5. **Audit file generation** - An audit file containing all necessary information to replicate and/or verify the election is generated.

Instant Runoff Algorithm

The first voting method this system follows is the Instant Runoff. This method works entirely with individual candidates, and will continuously move on to the next round until a winner is determined. The algorithm of this method can be summarized as follows:

1. Determine if the number of candidates who are still in the race.
    a. If there are two candidates and they are tied, flip a fair coin to decide the winner.
2. Determine if there is a candidate whose vote count accounts for 50% or more of the total number of votes.
    a. If there is one, that candidate will immediately be declared as the winner.
    b. If there is none, the candidate(s) with lowest number of votes will be eliminated, and their votes will be transferred to the candidates of the next preferences of their voters.
3. Recount the number of votes each candidate (excluding the eliminated candidate(s)).
4. Return to step 1.

Closed Party List Algorithm

Another voting method that this system follows is the Closed Party List. This method involves working with parties as a whole, rather than individual candidates. The algorithm of this method can be summarized as follows:

1. Calculate the quota by dividing the total number of ballots by the total number of seats
2. Divide the number of votes each party has by the quota, round the result down to the nearest whole number, allocate the seats corresponding to the calculated numbers to them. During the allocation, if a party runs out of available candidates, the remaining seats will be equally allocated to any of the other parties.
3. Once the first allocation is done, the number of votes each party receives will be recounted by subtracting the original number by the product of the quota and the number of seats that has been allocated to them.
4. Recount the total number of seats.
5. Equally allocate the remaining seats to the parties with the most vote counts. The logic within this procedure will be the same as step 2.

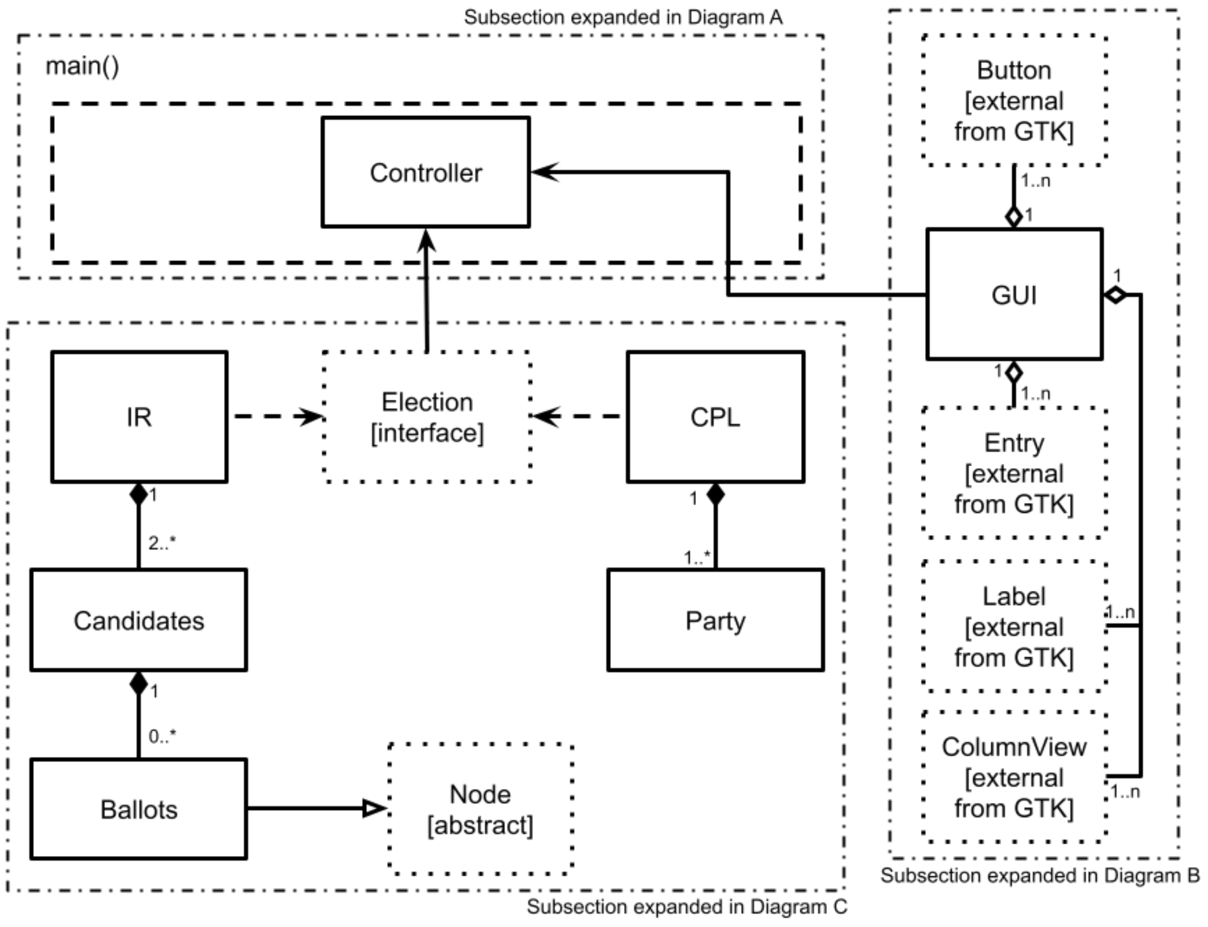
3. SYSTEM ARCHITECTURE

3.1. Architectural Design

The highest level of our proposed architecture is a class named Controller which is instantiated in the main() call of the program. This class handles interaction between two subgroups of classes -those relating to the graphical user interface and those relating to the logical processing of ballot data to produce election results. The class diagram below shows the interaction between each class in these groups.

Within the election processing subgroup, the strategy programming pattern will be used, where an Election class serves as an interface which is inherited by IR and CPL, implementation classes. This allows for a standardized interaction with the Controller class, while accounting for the significant differences between the election types. The two child classes, IR and CPL, have their own unique rules and logic to determine the winning candidate (for IR) or to determine the number of seats assigned to each political party (for CPL). Also, they contain different sets of classes for data storage and processing, as shown in the following diagrams. Details of data structure are in section 4.1

Note that the first diagram shows the entire system, while omitting class details in the interest of space. Following diagrams expand to the full UML class diagram level of detail. The detailed diagrams are labeled in the overall diagram.


Subsection expanded in Diagram A
main()
Controller
Button [external from GTK]
1..n
1
GUI
1
1
1..n
Entry [external from GTK]
Label [external from GTK]
1..n
ColumnView [external from GTK]
1..n
Subsection expanded in Diagram B
IR
Election [interface]
CPL
1
2..*
Candidates
1
1..*
Party
1
0..*
Ballots
Node [abstract]
Subsection expanded in Diagram C

Details of each class are shown in the following diagrams:

Diagram A:

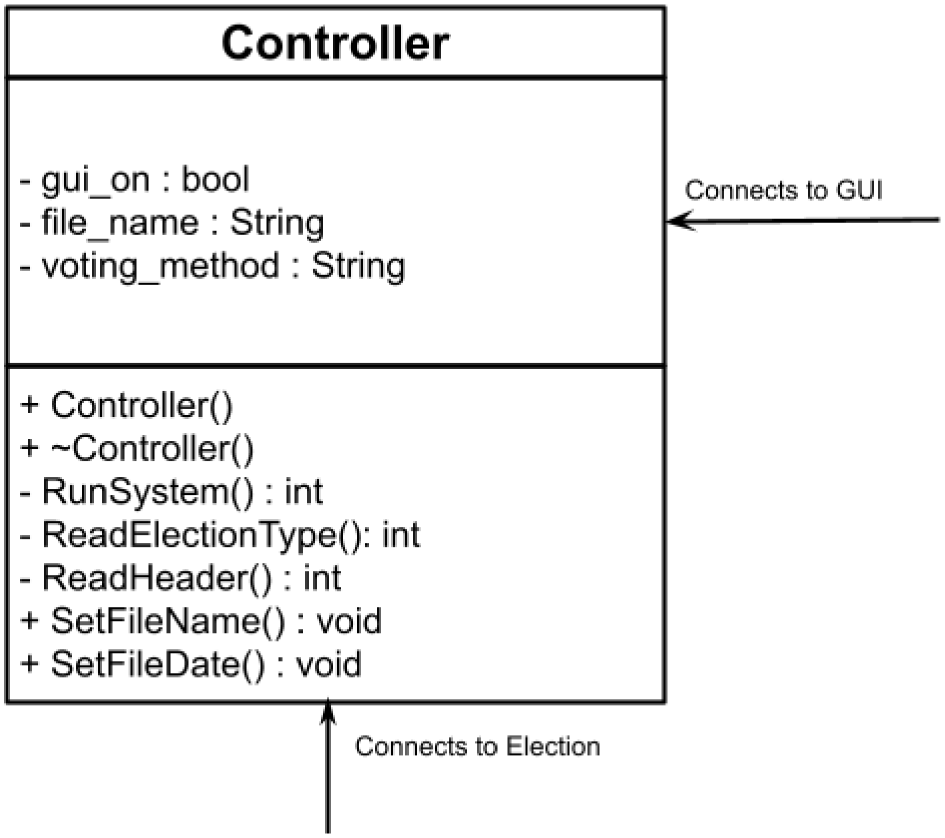


Diagram B:

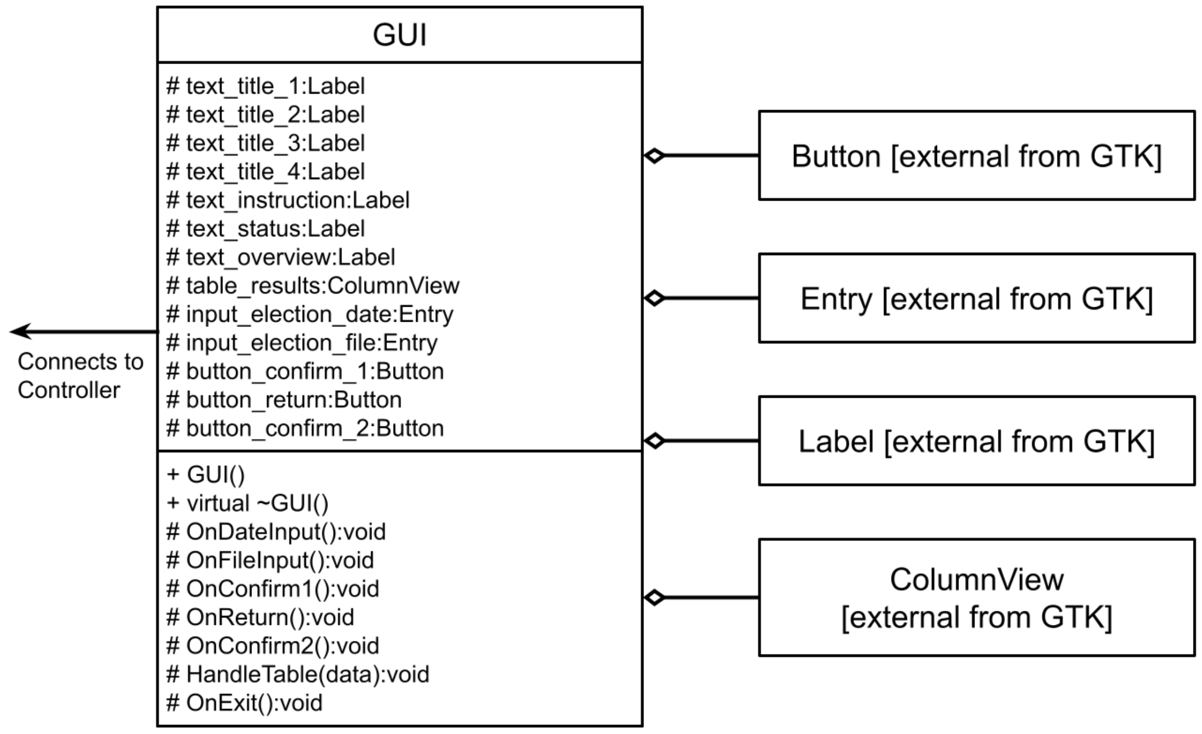
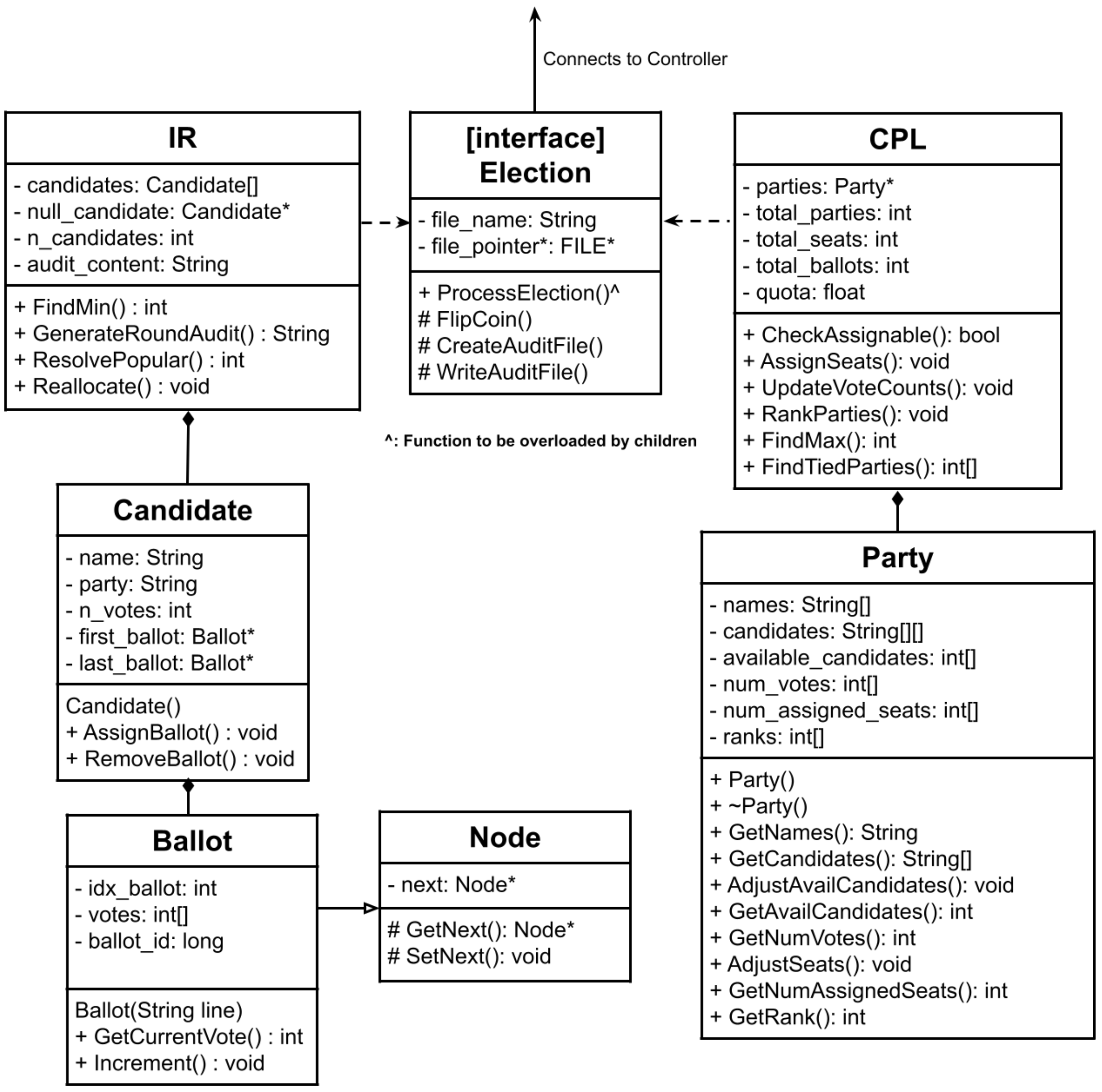
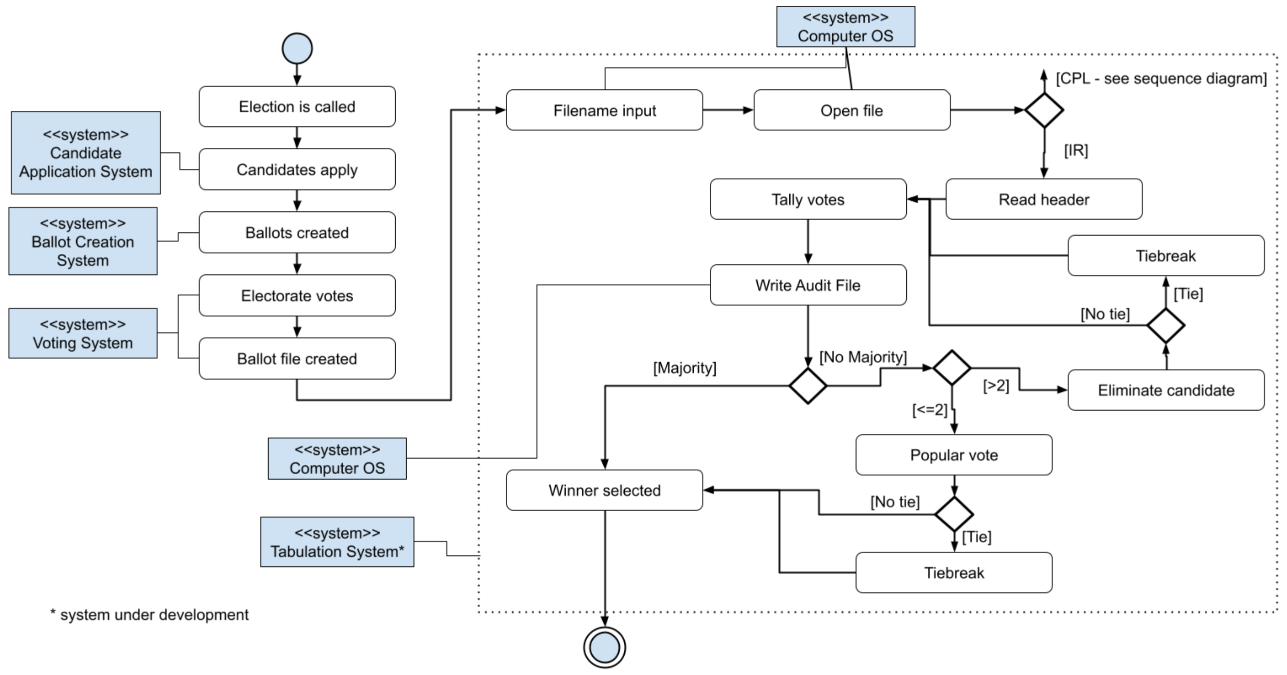

GUI
# text_title_1:Label
# text_title_2:Label
# text_title_3:Label
# text_title_4:Label
# text_instruction:Label
# text_status:Label
# text_overview:Label
# table_results:ColumnView
# input_election_date:Entry
# input_election_file:Entry
# button_confirm_1:Button
# button_return:Button
# button_confirm_2:Button
+ GUI()
+ virtual ~GUI()
# OnDateInput():void
# OnFileInput():void
# OnConfirm1():void
# OnReturn():void
# OnConfirm2():void
# HandleTable(data):void
# OnExit():void
Connects to Controller
Button [external from GTK]
Entry [external from GTK]
Label [external from GTK]
ColumnView [external from GTK]

Diagram C:



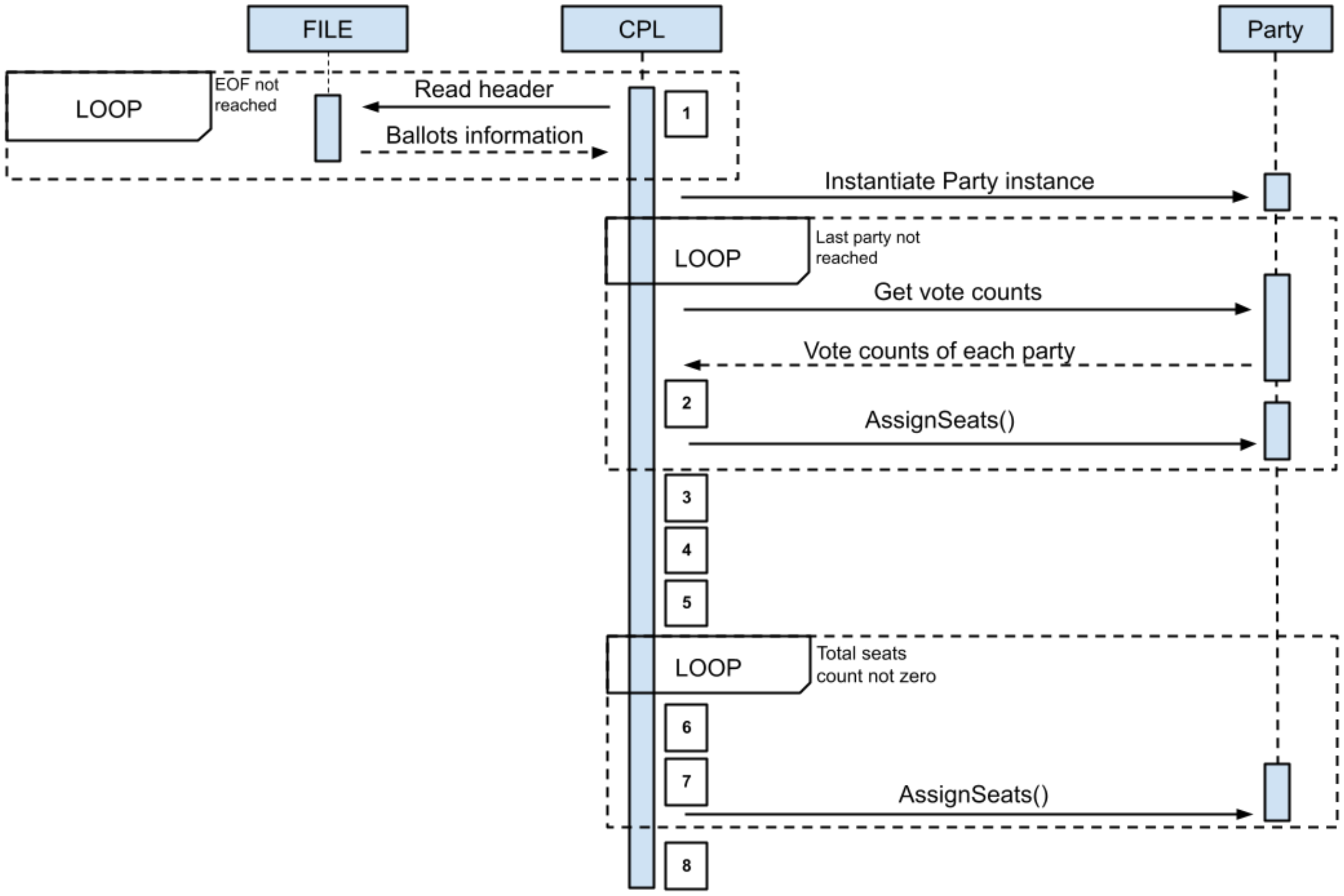
3.2. Decomposition Description

The following diagram is an activity diagram of the Instant Runoff portion of the election tabulation system. The diagram shows the steps from an election being called to the determination of a winner. Points where systems interact with process steps are identified, including both external systems and the ballot tabulation system described by this document. There is a step in the process where activities branch between IR and CPL. The CPL branch is described in the following sequence diagram and is therefore omitted.



The following sequence diagram gives a general view of how the CPL process works and what classes are interacting with each other throughout the process. The diagram predicts the timestamp after the results file has already been brought into the system, from the point of gathering information from the files and storing it, to calculating necessary parameters, to allocating seats to the parties, and to creating an audit file showing the information of the parties in terms of how the seats are allocated to them over two attempts. The following ideas will give more information about the diagram along with the arrows:

1. Read header and store the given information into CPL attributes and temporary arrays for future instantiation of Party object
2. Calculate seats amount to assign
3. Update seats count after first assignment
4. Call RankParties function to rank the parties based on their remaining vote counts
5. Keep track of the number of parties per rank
6. Check availables candidates of current rank
7. If not enough, move on to next rank
8. Randomly choose a party of currently rank
9. Write into audit file



3.3. Design Rationale

There was considerable deliberation about the structuring of the election classes, and how they would interact with both the controller and the actual GUI. In the end, it was decided that the controller would be the core class to handle top-down functionality, and that we would use the Strategy Pattern to best handle the different kinds of elections. In employing the Strategy pattern, we create a general election interface which has references for all key functions of elections. The controller is able to make these calls to the instantiated interface generically, without distinction between which kind of election is being employed. The election types themselves are specified on instantiation of the interface. This provides for a clear functional flow and minimizes the impact of changing details of election type implementation and adding any potential future election types.

The GUI then primarily interacts with the Controller class; Controller passes data received from the election interface to GUI as needed, and follows instructions/resolves inputs from GUI. This reduces the need to directly connect more classes (for example, if GUI needed to connect directly with the election instance AND Controller to provide functionality) and allows for a degree of separation between the GUI and the actual functionality of the code (which is useful, considering the existence of the non-GUI command line mode).

Another decision of importance is the splitting of election handling at all. Feasibly both types could be processed from the same class. This would provide considerable difficulty to testing, bug fixing, and any sort of future feature addition or alteration, however, which is not ideal. As such, the methods for each election were split, and then even more they were shaped to the Strategy Pattern as described above.

We had many choices for data and variable types with this project, so here are a few of the decisions made and why:

- Default "int" - At worst on a modern system, this is a 32 bit signed integer, which reaches a number well in excess of any nation's population on Earth today (approximately 2.1 billion). These numbers are sufficiently large.
- IR uses linked lists for certain things that CPL uses standard arrays for. This is intentional and has to do with the data scope of each election type. CPL does not need as elaborate a data type to handle many of its details, as there is no instant runoff process to contend with that expands the details of a ballot.

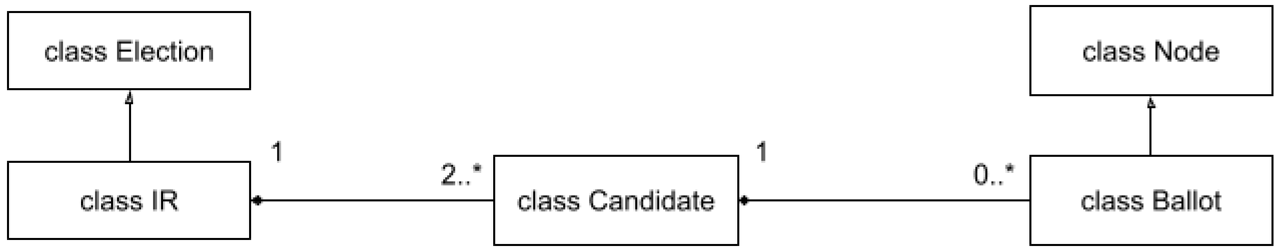
4. DATA DESIGN

4.1. Data Description

The primary data structure of this system is storing ballot information. Due to significant differences in the election process, data will be stored differently for each election type. Note that some variables describe indexes for ballots and candidates. These variables are integers, which means that the maximum number of ballots or candidates will be 2^{31} -1, or roughly 2 billion. This is significantly larger than the population of the United States.

Instant Runoff

The primary data structure for Instant Runoff implementation will have two tiers, as shown in the following diagram. Within the IR election class will be an array of pointers to instances of the class Candidate. Each candidate will contain a singly linked list of ballot objects. This form of storage is efficient in two senses. The size of the linked list for each candidate adapts to the number of ballots and the memory does not need to be preallocated. Also, the process of moving ballot data from one candidate to another can be done by reassigning pointers, and does not require further manipulation of the data itself.



The classes IR, Candidate, and Ballot include relevant attributes described in section 4.2. Note that the attributes not related to ballot data storage are omitted for brevity.

Closed Party List

Unlike the Instant Runoff method, Closed Party List method does not require ballots to be handled in a high-level way. Since the number of parties remains unchanged, the ballot elements will also remain unchanged, even though their tallies can still be adjusted. As such, it is not necessary for this approach to utilize a dedicated Ballot class to store the ballots information. Therefore, a data structure allowing easy access and modification like an array is more than sufficient for this implementation.

4.2. Data Dictionary

1. Class `Ballot`

ATTRIBUTE NAME	ATTRIBUTE TYPE	NOTE
`ballot_id`	`int`	Index of ballot in ballot file
`idx_ballot`	`int`	Index of current active vote
`votes`	`int[]`	Content of a ballot

2. Class `Candidate`

ATTRIBUTE NAME	ATTRIBUTE TYPE	NOTE
`first_ballot`	`Ballot*`	Pointer to first ballot in linked list
`last_ballot`	`Ballot*`	Pointer to last ballot in linked list
`n_votes`	`int`	Number of votes a candidate has
`name`	`String`	Name of a candidate
`party`	`String`	Political party a candidate belongs to

3. Class `CPL`

ATTRIBUTE NAME	ATTRIBUTE TYPE	NOTE
`parties`	`Party*`	Pointer to an instance of Party class
`quota`	`float`	The quota ratio determined by dividing the total vote counts by the number of available seats
`total _parties`	`int`	Number of parties
`total_ballots`	`int`	Number of ballots received
`total_seats`	`int`	Number of available seats

4. Class `IR`

ATTRIBUTE NAME	ATTRIBUTE TYPE	NOTE
`candidates`	`Candidate[]`	Array of pointers to candidate instances
`n_candidates`	`int`	
`null_candidate`	`Candidate*`	Pointer to candidate instance for storing ballots which no longer belong to any

5. Class `Party`

ATTRIBUTE NAME	ATTRIBUTE TYPE	NOTE
`available_candidates`	`int[]`	Number of free candidates of each of the parties
`candidates`	`String[][]`	Names of the candidates of each of the parties
`names`	`String[]`	Names of each of the parties
`num_assigned_seats`	`int[]`	Number of seats that each of the parties is possessing
`num_votes`	`int[]`	Vote counts of each of the parties
`ranks`	`int[]`	Rank of each of the parties based on their vote counts

- See Section 5 for detailed description of functions and their implementations.

5. COMPONENT DESIGN

Controller

```
RunSystem()
    Handle main arguments (GUI mode status and possible file name entry)
    Enter user input stage if file name is not included
        Update GUI to user input stage if GUI is enabled
        Wait for user entry -> filename, date_info
    ReadElectionType()
    ReadHeader()
    Update interface (GUI if enabled) to verification stage
    If user confirms, update interface (GUI if enabled) to tabulation stage
```

```
     Instantiate Election class of correct election type
     <Election>.ProcessElection()
     On completion of election processing, update interface (GUI if enabled) to
     display results

ReadElectionType()
     Open file
     Read first line
     Parse into election type

ReadHeader()
     Read head file, using election type to determine line count

SetFileName(String name)
     Updates file_name to name

SetFileDate(String date)
     Updates file_name to have appropriate date
```

Election

```
FlipCoin(int num)
     srand(time(0))
     Return (rand() % (num + 1)) - 1

CreateAuditFile()
     file_pointer = fopen("file_name")

WriteAuditFile()
     Write the information saved in class Election into the file
```

Instant Runoff Strategy

```
ProcessElection()
     // Read all ballot files into class
     ballot* temp
     While (lines exist)
          Line = read_line()
          Temp = ballot(line)
          idx_cand = temp.GetCurrentVote()
          candidates[idx_cand].AssignBallot(temp)


     // Begin round loop
```

```
          // Each round includes
          // check break cases
          If (candidate has majority)
               Break and set winner
          Else if (candidate count = 2)
               Break and go to popularity vote

          // find_min
          Idx_min = FindMin()

          // write audit
          Audit_string = GenerateRoundAudit()
          WriteAuditFile(audit_string)

          // reallocate
          Reallocate()

ResolvePopular(int idx_1, int idx_2) : int index
     Compare vote counts
     If tied, call BreakTie()
     Return index of winner

FindMin()
     Min = candidate_1
     For candidate in candidates
          If (candidate.ballot_count < min.ballot_count ) &&
(candidate.ballot_count > 0)
               Min = candidate
     Return min.index

Reallocate(idx_rm)
     For ballot in candidates[idx_rm].ballots
          idx_new = ballot.increment()

          // Move ballot to new candidate

          candidate[idx_new].assign_ballot(ballot)

          candidate[idx_rm].remove_ballot(idx_ballot)

AssignBallot(ballot)

     last_ballot.set_next(ballot)

     Last_ballot = ballot

     Ballot.set_next = null

RemoveBallot(ballot)
```

```
    ballot* prev, next
    prev = ballot->prev;
    prev->next = ballot->next;
```

Closed Party List (CPL)

```
ProcessElection()
    // Function containing the main logic of CPL
    // Initial round
    Loop through each party:
        Get vote counts and calculate the number of seats this party will be
        assigned
        Assign the seats to this party by using function AssignSeats
        // In AssignSeats function:
        //   Party information is updated
        //   Tie checking and coin flipping happen multiple times if this
    party runs out of available candidates
        //   Assign all the remaining seats to other parties following the
    previous step
        Update the total number of seats

    // Remainder round
    Rank the parties based on their remaining number of votes
    Loop until the total number of seats reaches zero:
        Find all the parties that have the same rank
        If no parties in one rank have enough candidates for seats
    assignment:
            Move to the next rank
        Flip the coin to choose a lucky party among the said parties
        Assign the seats to this party using function AssignSeats
        // The steps in this function will be the same as in initial round
    Return

GetQuota()
    Return the quota

AssignSeats()
    // Helper function used in ProcessElection function to assign seats to
    given party
    Loop until the given number of seats reaches zero:
        Verify availability of candidates using CheckAssignable function
        If available:
            Update the number of seats this party possesses
            Update the number of available candidates of this party
        Else:
            Flip the coin to assign the seats to the another party
            Update the given number of seats
        Update the total number of seats
    Return

CheckAssignable()
    // Helper function used in AssignSeats function
    Return whether this party has enough candidates for seats assignment

UpdateVoteCounts()
    // Helper function used in ProcessElection function to modify and store
    the latest number of votes of each party
    Loop through each party:
        Subtract the vote counts of this party by the product of the quota
        and the number of seats allocated
    Return

RankParty()
    // Helper function used in ProcessElection function to find rank the
    parties
    upper_bound = infinity
    rank_index = 0
    desired_rank = 1
    Loop until reaching the end of order_counts array:
        Set value of upper_bound to be the value returned by FindMax function
        Loop through the vote counts array:
            Get indexes of the parties that have the same value as
            upper_bound
            Fill in the cells at those indexes in the ranks array with the
            value of desired_rank
    Return

FindMax()
    largest = vote counts array at the first index
    Loop through the vote counts array:
        If the vote counts at index is larger than at largest but smaller
        than given upper bound:
            Set largest to be index
    Return vote counts at largest

FindTiedParties()
    // Helper function used in ProcessElection function to find all the
    parties that are tied
    Loop through the array containing the numbers of votes:
        If the vote counts at this index is equal to a desired value:
            Add this index to the array containing all the parties that are
            tied
            Increase the number of tied parties
    Return
```

GUI

```
GUI Class inheriting from Gtk::Window
        public:
                GUI()
                        Initialize all Gtk elements
                        Configure GUI settings (border with, etc.)
                        Connect input_election_date with on_date_input()
                        Connect input_election_file with on_file_input()
                        Connect button_confirm_1 with on_confirm_1()
                        Connect button_return with on_return()
                        Connect button_confirm_2 with on_confirm_2()
                        Connect table_resutls with handle_table()
                        Add first elements into container
                virtual ~GUI()

        Protected:
                Gtk::Label text_title_1
                Gtk::Label text_title_2
                Gtk::Label text_title_3
                Gtk::Label text_title_4
                Gtk::Label text_instruction
                Gtk::Label text_status
                Gtk::Label text_overview
                Gtk::ColumnView table_results
                Gtk::Entry input_election_date
                Gtk::Entry input_election_file
                Gtk::Button button_confirm_1
                Gtk::Button button_return
                Gtk::Button button_confirm_2

                void OnDateInput()
                        Signals Controller to handle inputted data

                void OnFileInput()
                        Signals Controller to handle inputted data

                void OnConfirm1()
                        Signals Controller to handle confirmation
                void OnReturn()
```

```
          Signals Controller to handle return

     void OnConfirm2()
          Signals Controller to handle confirmation

     void HandleTable(data)
          Parses data into table_results

     void OnExit()
          Handles exiting of program to gently close audit file
```

6. HUMAN INTERFACE DESIGN

6.1. Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

- There will be two "modes" for the software:
    - Standard mode: contains a GUI made using GTK, and is accessible for all users.
    - Maintenance mode: contains exclusively command line interface, and is manually accessible by the programmers and testers on launch with command line arguments.
- The interface for both will need to provide visual output and at designated times take user input.
- Any windows along the way generated for the GUI will have standard OS controls (minimize and close) which will be handled appropriately.
- The command line interface will provide similar functionality to the GUI, but any displayed content will be plaintext, and any inputs will be entered via the command line.
    - File path entry becomes a command line entry.
    - Confirmation buttons become (y)es/(n)o text entries.

Details of the GUI:

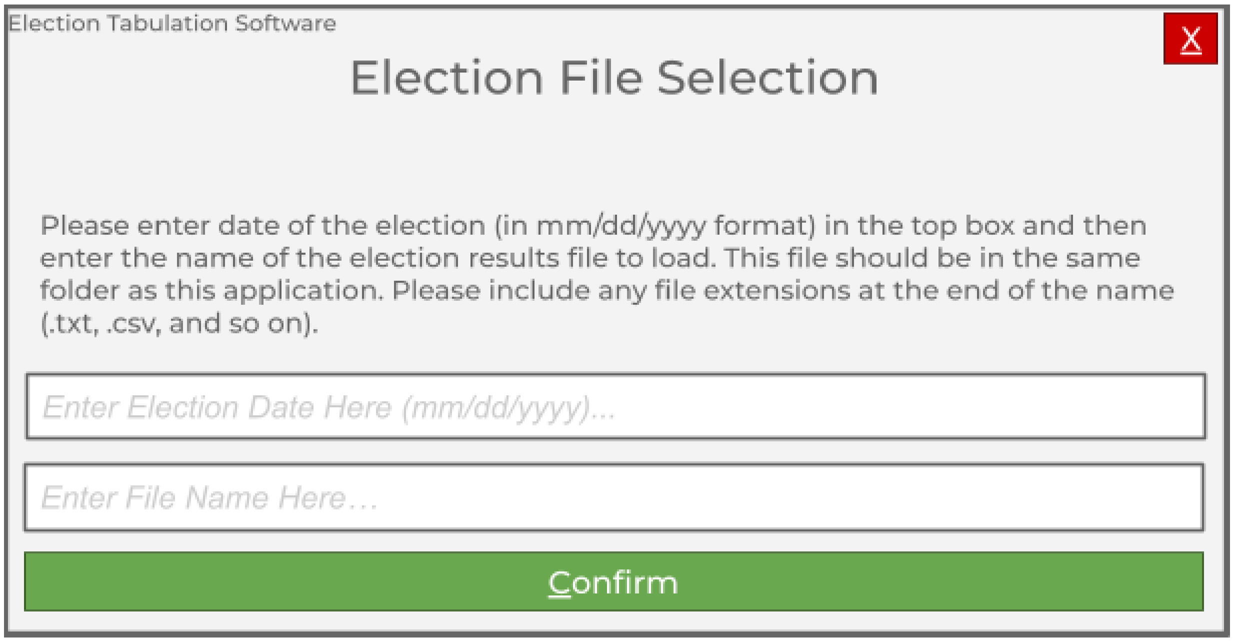
1. Once the system is launched, there will be two text entry boxes for election date (in mm/dd/yyyy format) and file name. Entering the file path to the desired file and clicking "Confirm" (shortcut *c*) will bring the user to the next interface. (*see Figure 1*)

2. After the user enters and confirms the file path, a small information text box containing the header information of the file will be shown to the user, or an error message will be displayed if something is wrong. The user can confirm the validity of the file (shortcut *C*) or revert back to the previous interface (shortcut *B*) if they opened the wrong file. If confirmed, the system will proceed to the next step and direct the user to the next interface. (*See Figure 2*)

3. In this interface, the system will notify the user that it is tabulating the ballots. (*See Figure 3*)

4. When tabulation is complete, it will display a results screen showing certain information as text to the user. (*See Figure 4*)

6.2. Screen Images



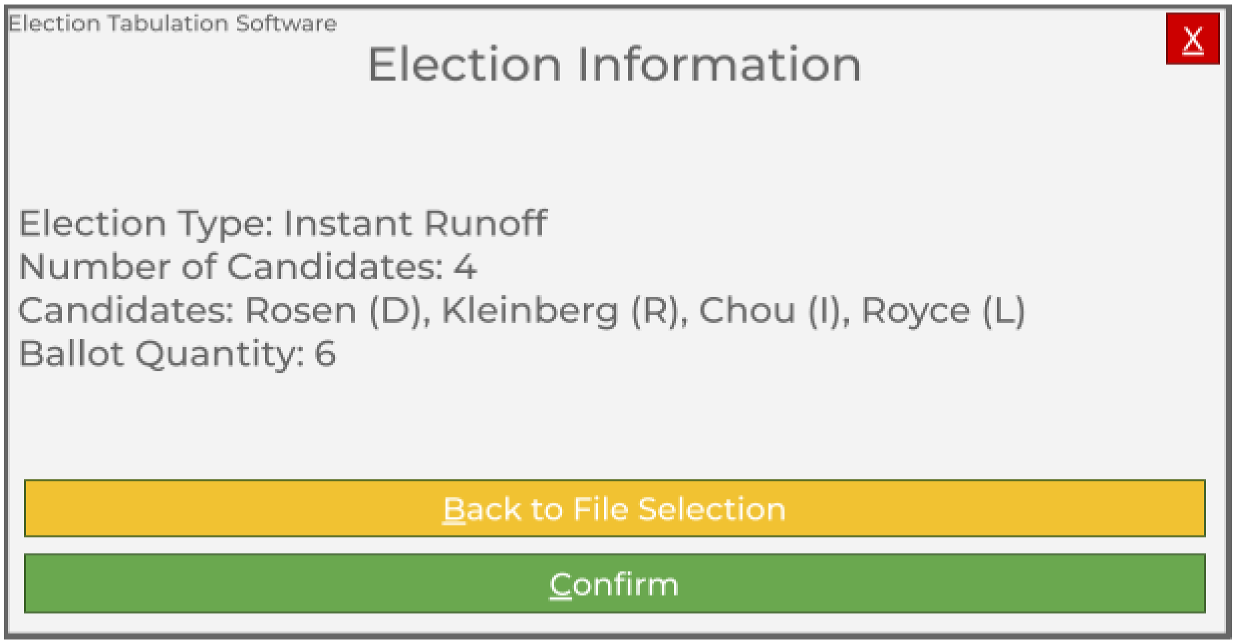
Election File Selection

Please enter date of the election (in mm/dd/yyyy format) in the top box and then enter the name of the election results file to load. This file should be in the same folder as this application. Please include any file extensions at the end of the name (.txt, .csv, and so on).

Enter Election Date Here (mm/dd/yyyy)...

Enter File Name Here…

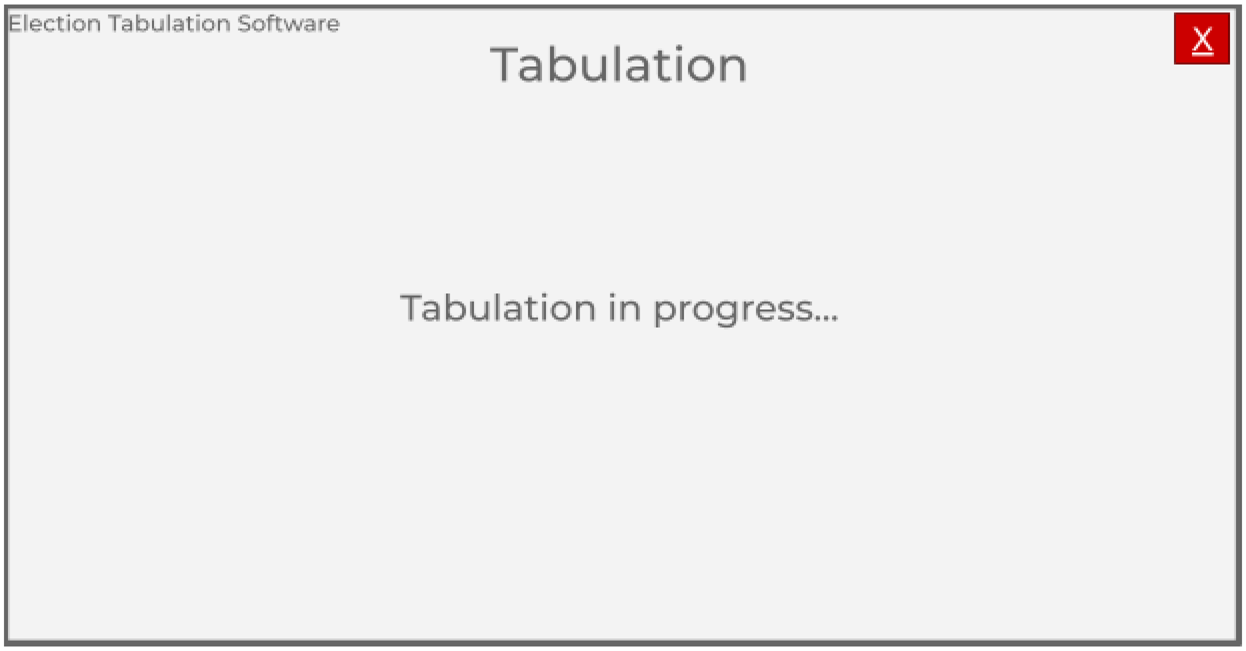
(Figure 1)



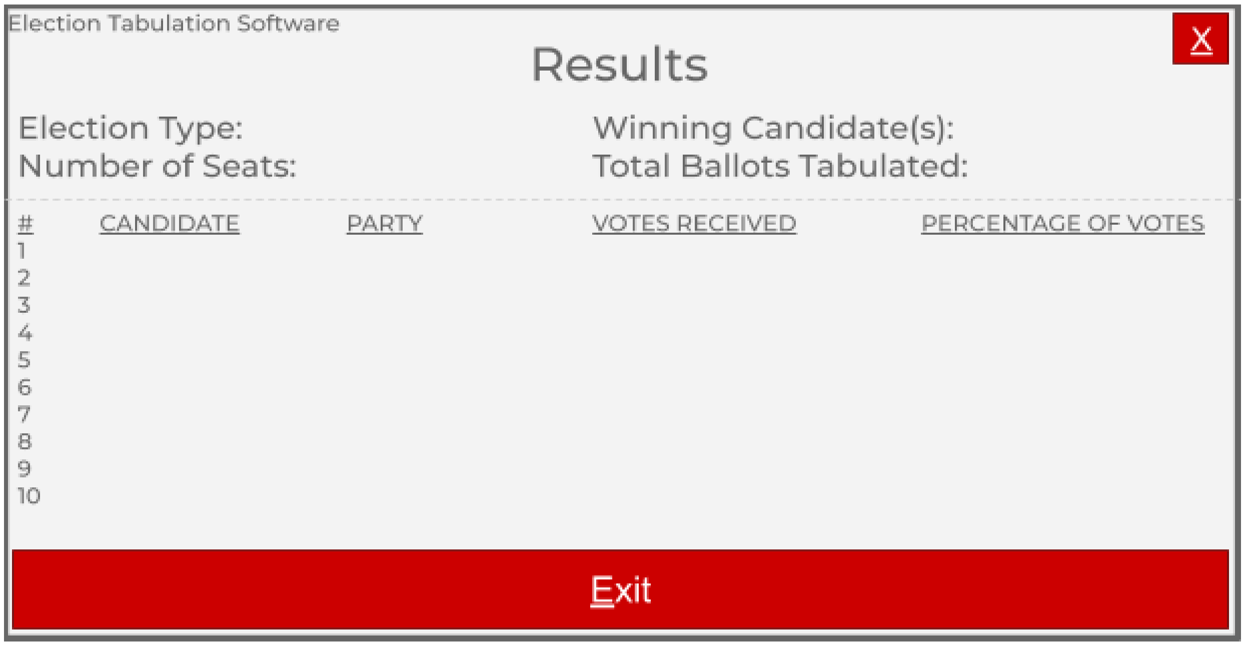
Election Information

Election Type: Instant Runoff
Number of Candidates: 4
Candidates: Rosen (D), Kleinberg (R), Chou (I), Royce (L)
Ballot Quantity: 6

(Figure 2)



(Figure 3)



Election Type:
Number of Seats:
Winning Candidate(s):
Total Ballots Tabulated:

#	CANDIDATE	PARTY	VOTES RECEIVED	PERCENTAGE OF VOTES
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

(Figure 4)

6.3. Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.

These objects only exist for the GUI. The command line mode will only feature text for user IO and as such would not have graphical "objects".

Breakdown of the GUI:

- The most basic screen object that will be present and unchanged for the entire operation of the software is the background window/panel itself.
    - The panel will utilize standardized OS control features such as the “close” button, minimize, etc. - this is automatic to the environment and tools used, and do not require our explicit implementation.
- There will be four different window layouts (*see figures 1-4*) for the different “phases” of the software.
- Each layout will have title text indicating the phase of the software (Election File Selection, Election Information, Tabulation, and Results).
- Layout 1 will additionally feature:
    - A text display box with instructions for the user.
    - A text input entry box for the user to enter the election date.
    - A text input entry box for the user to enter the election file name.
    - A button for confirmation of information. When pressed, the text in each entry box will be scanned and processed for further progression to Layout 2.
- Layout 2 will additionally feature:
    - A text display box with information about the selected election file.
    - A button to return to the file selection page (Layout 1).
    - A button for user confirmation which proceeds to Layout 3.
- Layout 3 will additionally feature:
    - A text display box with a message that tabulation is in progress. There is no user input to this Layout; when calculation is complete, the system will automatically move to Layout 4.
- Layout 4 will additionally feature:
    - A text display box with overview election information such as the election type, winning candidate(s), number of seats for the election, and total ballots tabulated.
    - A text display table which features the results for each candidate and/or party on a sorted table, with details such as place in the election, candidate name, party name, votes received, and percentage of total votes said votes account for.

7. REQUIREMENTS MATRIX

Provide a cross reference 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.

SYSTEM COMPONENTS	FUNCTIONAL REQUIREMENTS	ID
- Controller	Load ballot file	UC_001
- Controller	Read ballot file header	UC_002
- IR - CPL	Process election	UC_003

- IR - CPL	Break a tie	UC_004
- Controller - GUI	Display result page	UC_005
- Election	Create audit file	UC_006
- IR	Tabulate IR	UC_007
- CPL	Tabulate CPL	UC_008
- Election	Name audit file	UC_009
- IR	Resolve inconclusive IR as popularity vote	UC_010
- GUI - Controller	Extract additional information from the user	UC_011
- IR - CPL	Read ballot file	UC_012
- Controller	File name input as command line argument	UC_013

8. APPENDICES

Nothing of note added in this document, see the SRS for anything not covered here.