David Zhang
v0 ConveyorFamily Pseudocode Design

Notes/Conventions:

- Organization
  - The agents Conveyor, Popup, and Sensor are each in a ConveyorFamily class, and *they compose the ConveyorFamily to be demonstrated in v0*.
  - The agents Robot and Machine compose an entity called the Workstation, which has stubbed methods for interaction (not critical in v0 - they are not part of the insides of this design of the ConveyorFamily)
- Naming
  - All msgXX (where XX is a message name in the interaction diagram) are messages
  - All doYY methods are gui/animation-related/transducer methods
  - All actZZ are the actions called in the scheduler

Conveyor

Conveyor Data

```
private ConveyorFamilyEntity family;
        private Transducer t;
        public enum ConveyorState { GLASS_JUST_ARRIVED,
WAITING_FOR_GLASS_TO_REACH_ENDING_SENSOR,
SHOULD_NOTIFY_POSITION_FREE, NOTHING_TO_DO }
        private ConveyorState state = ConveyorState.NOTHING_TO_DO;

        private List<Glass> glasses = Collections.synchronizedList(new ArrayList<Glass>());


// Constructor
public ConveyorAgent(ConveyorFamilyEntity f, Transducer transducer) {
                family = f;
                t = transducer;
                t.register(this, TChannel.SENSOR);
        }
```

Conveyor Messages

```
public void msgHereIsGlass(Glass g) {
                state = ConveyorState.GLASS_JUST_ARRIVED; // previous sensor should have
already started the conveyor
                // at this point, this should be true: family.runningState ==
RunningState.ON_BC_SENSOR_TO_CONVEYOR
```

```
            glasses.add(g);
            stateChanged();
        }


        public void msgTakingGlass() {
            state = ConveyorState.SHOULD_NOTIFY_POSITION_FREE;
            glasses.remove(0);
            stateChanged();
        }
```

Conveyor Scheduler

```
if (state == ConveyorState.GLASS_JUST_ARRIVED) {
                        // !glasses.isEmpty() should be true
                        state =
ConveyorState.WAITING_FOR_GLASS_TO_REACH_ENDING_SENSOR;
                        actTellPopupGlassOnConveyor(glasses.get(0));
                        return true;
                } else if (state ==
ConveyorState.WAITING_FOR_GLASS_TO_REACH_ENDING_SENSOR) { // technically could
be merged into NOTHING_TO_DO
                        // Do nothing. Next thing that happens is conveyor auto-stops via
eventFired, popup agent realizes sensorOccupied = true,
                        // does actLoadGlassOntoPopup which *then tells this conveyor agent
msgTakingGlass()*
                        return false;
                } else if (state == ConveyorState.SHOULD_NOTIFY_POSITION_FREE) {
                        state = ConveyorState.NOTHING_TO_DO;
                        actTellSensorPositionFree();
                        return false;
                } else { // NOTHING_TO_DO
                        return false;
                }
```

Conveyor Actions

```
public void actTellPopupGlassOnConveyor(Glass g) {
                GlassState glassState = family.decideIfGlassNeedsProcessing(g); // conveyor
decides this since it has time
                MyGlass myGlass = family.new MyGlass(g, glassState);

                family.runningState = RunningState.ON_BC_CONVEYOR_TO_SENSOR;
```

```
                family.popup.msgGlassComing(myGlass);

                // Trust that conveyor knows to stop glass the moment the right sensor fires. See
eventFired.
        }

        public void actTellSensorPositionFree() {
                family.sensor.msgPositionFree();
        }
```

Popup

Popup Data

```
private ConveyorFamilyEntity family;
        private Transducer t;
        private Workstation workstation1; // top workstation, higher priority, one with lower index
        private Workstation workstation2; // bottom workstation
        private TChannel workstationChannel;
        private List<MyGlass> glasses = Collections.synchronizedList(new
ArrayList<MyGlass>()); // uses MyGlass instead of just Glass so it contains GlassState
        // A glass is removed from glasses when it is messaged to a workstation. Then,
workstation eventually sends glass back,
        // and the glass is added to finishedGlasses.
        private List<Glass> finishedGlasses = Collections.synchronizedList(new
ArrayList<Glass>());
        private boolean nextPosFree = false;
        private boolean sensorOccupied = false; // roughly equivalent to
family.runningState.OFF_BC_WAITING_AT_SENSOR, but needed for popup to internally decide
to move to ON_BC_SENSOR_TO_POPUP
        private boolean isUp = false; // up or down; starts out down

        // Mainly used to differentiate between waiting for a transducer event to fire (WAIT_FOR)
and when popup should actually check scheduler events (ACTIVE)
        // if (ACTIVE) is used in scheduler, if (WAIT_FOR_SOMETHING) is used in eventFired to
signal the popup is DOING_NOTHING for some animation to finish
        public enum PopupState { ACTIVE,
                WAITING_FOR_LOW_POPUP_BEFORE_LOADING_TO_WORKSTATION,
                WAITING_FOR_HIGH_POPUP_BEFORE_LOADING_TO_WORKSTATION,

WAITING_FOR_HIGH_POPUP_BEFORE_RELEASING_FROM_WORKSTATION,
                WAITING_FOR_LOW_POPUP_WITH_GLASS_FROM_WORKSTATION,
                WAITING_FOR_LOW_POPUP_BEFORE_RELEASE,
                WAITING_FOR_GLASS_TO_COME_FROM_SENSOR_BEFORE_RELEASING,
```

```
WAITING_FOR_GLASS_TO_COME_FROM_SENSOR_BEFORE_LOADING_TO_WORKSTATION,
		WAITING_FOR_WORKSTATION_GLASS_RELEASE, DOING_NOTHING
		} // DOING_NOTHING is the default, doing nothing state - neither checking scheduler nor waiting for an animation to finish
	public enum WorkstationState { FREE, BUSY, DONE_BUT_STILL_HAS_GLASS }
	PopupState state = PopupState.DOING_NOTHING;
	WorkstationState wsState1 = WorkstationState.FREE;
	WorkstationState wsState2 = WorkstationState.FREE;
```

Popup Messages

```
public void msgGlassComing(MyGlass myGlass) {
		glasses.add(myGlass);
		if (state == PopupState.DOING_NOTHING) {
			state = PopupState.ACTIVE;
			stateChanged(); // only check scheduler if doing nothing
		}
	}


	public void msgPositionFree() {
		nextPosFree = true;
		if (state == PopupState.DOING_NOTHING) {
			state = PopupState.ACTIVE;
			stateChanged(); // only check scheduler if doing nothing
		}
	}


	public void msgGlassDone(Glass g, int machineIndex) {
		updateWorkstationState(machineIndex, WorkstationState.DONE_BUT_STILL_HAS_GLASS);
		finishedGlasses.add(g);

		if (state == PopupState.DOING_NOTHING) {
			state = PopupState.ACTIVE;
			stateChanged(); // only check scheduler if doing nothing
		}
		// otherwise, popup is busy WAITING_FOR something else to happen, or is already ACTIVE doing something perhaps for the other workstation
	}
```

Popup Scheduler

```
        // ACTIVE is set by transducer and incoming messages. We only take action if we are 'active'.
                if (state == PopupState.ACTIVE) {
                        // Case 1 (easy): Just deal with the workstation's finished glass by passing it on. No complications with sensor.
                        if (!sensorOccupied) {
                                // If next position is free and there exists a glass in glasses list that is finished by a workstation
                                if (nextPosFree && atLeastOneWorkstationIsDoneButStillHasGlass()) {
                                        // Keep state as ACTIVE. This is implied.
                                        actReleaseGlassFromWorkstation();
                                        return false;
                                }
                        }
                        // Case 2-x deal with when sensor is occupied, which adds complications.
                        else {
                                MyGlass g = getNextUnhandledGlass(); // the *unhandled* glass - we make the glass at the sensor more important than any glass at a workstation
                                if (g != null) { // should be present since sensorOccupied = true
                                        // Case 2: Regardless of workstation, just load sensor's glass and pass it on - no workstation interaction
                                        if (nextPosFree && !g.needsProcessing()) {
                                                actLoadSensorsGlassOntoPopupAndRelease();
                                                return false;
                                        }
                                        // Case 3: Release workstation's finished glass to next family
                                        else if (g.needsProcessing() && bothWorkstationsOccupiedButAtLeastOneIsDone() && nextPosFree) {
                                                actReleaseGlassFromWorkstation();
                                                return false;
                                        }
                                        // Case 4: Load sensor's glass onto workstation. Must happen after case 3 if case 3 happens.
                                        else if (g.needsProcessing() && aWorkstationIsFree()) {
                                                actLoadSensorsGlassOntoWorkstation();
                                                return false;
                                        }
                                } else {
```

```
                                System.err.println("Null unhandled glass!");
                        }
                }
        } // returning true above is actually meaningless since all act methods lead to
WAIT state, so we just reach false anyway.
        state = PopupState.DOING_NOTHING; // this could interfere with other wait
states if you returned true above
        return false;

public void eventFired(TChannel channel, TEvent event, Object[] args) {
        // Most checks here involve seeing if state is a form of WAITING_FOR, which
happen from scheduler actions.

        // Exception: we must update sensor status regardless of the state.
        if (!sensorOccupied) { // should only bother to check if sensor is not occupied -
here, the popup only cares about listening to see if a glass has arrived at the preceding sensor
                if (channel == TChannel.SENSOR && event ==
TEvent.SENSOR_GUI_PRESSED) {
                        // When the sensor right before the popup has been pressed, allow
loading of glass onto popup

                        // TODO: parse args to check if it is this sensor
                        state = PopupState.ACTIVE;
                        sensorOccupied = true;
                        stateChanged();
                }
        }

        // From actLoadSensorsGlassOntoWorkstation, step 2 (sometimes)
        if (state ==
PopupState.WAITING_FOR_LOW_POPUP_BEFORE_LOADING_TO_WORKSTATION) {
                if (channel == TChannel.POPUP && event ==
TEvent.POPUP_GUI_MOVED_DOWN) {
                        // TODO: parse args to check if it is this popup
                        state =
PopupState.WAITING_FOR_GLASS_TO_COME_FROM_SENSOR_BEFORE_LOADING_TO_
WORKSTATION;
                        doStartConveyor();
                        family.runningState =
RunningState.ON_BC_SENSOR_TO_POPUP;
                        family.conv.msgTakingGlass();
                }
        }
```

```
            // From actLoadSensorsGlassOntoWorkstation, step 3
            if (state ==
PopupState.WAITING_FOR_GLASS_TO_COME_FROM_SENSOR_BEFORE_LOADING_TO_
WORKSTATION) {
                if (channel == TChannel.POPUP && event ==
TEvent.POPUP_GUI_LOAD_FINISHED) {
                    // TODO: parse args to check if it is this sensor
                    // if so:
                    state =
PopupState.WAITING_FOR_HIGH_POPUP_BEFORE_LOADING_TO_WORKSTATION;
                    sensorOccupied = false;
                    family.runningState = RunningState.OFF_BC_QUIET;
                    doStopConveyor();
                    doMovePopupUp();
                }
            }
            // From actLoadSensorsGlassOntoWorkstation, step 4 (final)
            if (state ==
PopupState.WAITING_FOR_HIGH_POPUP_BEFORE_LOADING_TO_WORKSTATION) {
                if (channel == TChannel.POPUP && event ==
TEvent.POPUP_GUI_MOVED_UP) {
                    // TODO: parse args to check if it is this popup
                    state = PopupState.ACTIVE;
                    MyGlass g = glasses.remove(0); // first glass should be the one

                    Workstation w =
getWorkstationWithState(WorkstationState.FREE);
                    updateWorkstationState(w, WorkstationState.BUSY);
                    w.msgHereIsGlass(g.getGlass());

                    doLoadGlassOntoWorkstation(w.getIndex());
                    stateChanged();
                }
            }

            // From actReleaseGlassFromWorkstation step 2 (sometimes)
            if (state ==
PopupState.WAITING_FOR_HIGH_POPUP_BEFORE_RELEASING_FROM_WORKSTATION)
{
                if (channel == TChannel.POPUP && event ==
TEvent.POPUP_GUI_MOVED_UP) {
                    state =
PopupState.WAITING_FOR_WORKSTATION_GLASS_RELEASE;
```

```
                        doReleaseGlassFromProperWorkstation();
                    }
                }
                // From actReleaseGlassFromWorkstation step 3
                if (state == PopupState.WAITING_FOR_WORKSTATION_GLASS_RELEASE) {
                    if (channel == this.workstationChannel && event ==
TEvent.WORKSTATION_RELEASE_FINISHED) {
                        state =
PopupState.WAITING_FOR_LOW_POPUP_WITH_GLASS_FROM_WORKSTATION;
                        doMovePopupDown();
                    }
                }
                // From actReleaseGlassFromWorkstation step 4 (final)
                if (state ==
PopupState.WAITING_FOR_LOW_POPUP_WITH_GLASS_FROM_WORKSTATION) {
                    if (channel == TChannel.POPUP && event ==
TEvent.POPUP_GUI_MOVED_DOWN) {
                        state = PopupState.ACTIVE;

                        // Here we can send the next family the message. No need to
check POPUP_GUI_RELEASE_FINISHED b/c that is detected _after_ the next family's sensor
already gets the glass.
                        Glass glass = finishedGlasses.remove(0); // remove & return first
element
                        family.nextFamily.msgHereIsGlass(glass);

                        doReleaseGlassFromPopup();

                        stateChanged();
                    }
                }

                // From actLoadSensorsGlassOntoPopupAndRelease, step 2 (sometimes)
                if (state == PopupState.WAITING_FOR_LOW_POPUP_BEFORE_RELEASE) {
                    if (channel == TChannel.POPUP && event ==
TEvent.POPUP_GUI_MOVED_DOWN) {
                        // TODO: parse args to check if it is this popup
                        // if so:
                        state =
PopupState.WAITING_FOR_GLASS_TO_COME_FROM_SENSOR_BEFORE_RELEASING;
                        doStartConveyor();
                        family.runningState =
RunningState.ON_BC_SENSOR_TO_POPUP;
```

```
                }
            }
            // From actLoadSensorsGlassOntoPopupAndRelease, step 3 (final)
            if (state ==
PopupState.WAITING_FOR_GLASS_TO_COME_FROM_SENSOR_BEFORE_RELEASING) {
                if (channel == TChannel.POPUP && event ==
TEvent.POPUP_GUI_LOAD_FINISHED) {
                    // TODO: parse args to check if it is this popup
                    // if so:
                    state = PopupState.ACTIVE;
                    sensorOccupied = false;

                    // Here we can send the next family the message. No need to
check POPUP_GUI_RELEASE_FINISHED b/c that is detected _after_ the next family's sensor
already gets the glass.
                    MyGlass mg = glasses.remove(0); // should be first glass
                    family.nextFamily.msgHereIsGlass(mg.getGlass());

                    family.runningState = RunningState.OFF_BC_QUIET;
                    doStopConveyor();
                    doReleaseGlassFromPopup();

                    stateChanged();
                }
            }
        }
```

Popup Actions

```
public void actLoadSensorsGlassOntoPopupAndRelease() {
            if (isUp) {
                state =
PopupState.WAITING_FOR_LOW_POPUP_BEFORE_RELEASE;
                doMovePopupDown();
            } else { // popup already down
                state =
PopupState.WAITING_FOR_GLASS_TO_COME_FROM_SENSOR_BEFORE_RELEASING;
                family.runningState = RunningState.ON_BC_SENSOR_TO_POPUP;
                doStartConveyor();
            }
        }

        // Multi-step with eventFired
```

```
	public void actLoadSensorsGlassOntoWorkstation() {
		if (isUp) {
			state =
PopupState.WAITING_FOR_LOW_POPUP_BEFORE_LOADING_TO_WORKSTATION;

			doMovePopupDown();
		} else { // popup already down
			state =
PopupState.WAITING_FOR_GLASS_TO_COME_FROM_SENSOR_BEFORE_LOADING_TO_
WORKSTATION;
			family.runningState = RunningState.ON_BC_SENSOR_TO_POPUP;
			doStartConveyor();
			family.conv.msgTakingGlass();
		}
	}

	/**
	 * Releases glass from workstation to next conveyor family
	 */
	public void actReleaseGlassFromWorkstation() {
		// Note: popup must be up -> WORKSTATION_RELEASE_FINISHED ->
POPUP_GUI_LOAD_FINISHED (implied) ->
		// POPUP_GUI_MOVED_DOWN -> automatically moves on

		// Make sure gui is up first
		if (!isUp) {
			state =
PopupState.WAITING_FOR_HIGH_POPUP_BEFORE_RELEASING_FROM_WORKSTATION;
			doMovePopupUp();
		} else { // popup already up
			state =
PopupState.WAITING_FOR_WORKSTATION_GLASS_RELEASE;
			doReleaseGlassFromProperWorkstation();
		}
	}
```

Sensor

Sensor Data

```
	private ConveyorFamilyEntity family;
	private Transducer t;
	public enum SensorState { SHOULD_NOTIFY_POSITION_FREE, NOTHING_TO_DO,
GLASS_JUST_ARRIVED }
```

```
	private SensorState state = SensorState.NOTHING_TO_DO;

	private List<Glass> glasses = Collections.synchronizedList(new ArrayList<Glass>());
```

Sensor Messages

```
public void msgHereIsGlass(Glass g) {
		state = SensorState.GLASS_JUST_ARRIVED;
		glasses.add(g);
		stateChanged();
	}

	public void msgPositionFree() {
		state = SensorState.SHOULD_NOTIFY_POSITION_FREE;
		stateChanged();
	}
```

Sensor Scheduler

```
if (state == SensorState.GLASS_JUST_ARRIVED) {
				state = SensorState.NOTHING_TO_DO;
				// !glasses.isEmpty() should be true
				actPassOnGlass(glasses.remove(0)); // remove because sensor passes on immediately no matter what
				return false;
			} else if (state == SensorState.SHOULD_NOTIFY_POSITION_FREE) {
				state = SensorState.NOTHING_TO_DO;
				actTellPrevFamilyPositionFree();
				return false;
			} else { // NOTHING_TO_DO
				return false;
			}
```

Sensor Actions

```
public void actPassOnGlass(Glass g) {
		while (family.runningState != RunningState.OFF_BC_QUIET) { // only supports one glass at a time
			// Wait until conveyor is officially in the proper off state.
			// This should be very quick and is only here in the event that *right after* conveyor tells this sensor msgPositionFree and this sensor tells the previous family, that family sends the next glass.
```

```
        }
        doStartConveyor();
        family.runningState = RunningState.ON_BC_SENSOR_TO_CONVEYOR;
        family.conv.msgHereIsGlass(g);
}

public void actTellPrevFamilyPositionFree() {
        family.prevFamily.msgPositionFree();
}
```